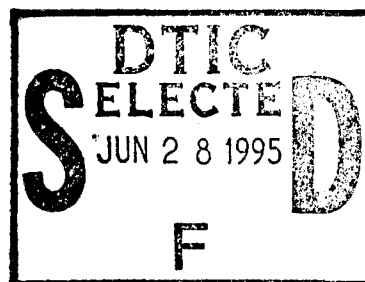


Center for Reliable and High Performance Computing



Fault-Sensitivity and Wear-Out Analysis of VLSI Systems

Gwan Seung Choi

19950626 067

*Coordinated Science Laboratory
College of Engineering*

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified			1b. RESTRICTIVE MARKINGS None		
2a. SECURITY CLASSIFICATION AUTHORITY			3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited		
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE					
4. PERFORMING ORGANIZATION REPORT NUMBER(S)			5. MONITORING ORGANIZATION REPORT NUMBER(S)		
6a. NAME OF PERFORMING ORGANIZATION Coordinated Science Lab University of Illinois		6b. OFFICE SYMBOL (If applicable) N/A		7a. NAME OF MONITORING ORGANIZATION National Aeronautics & Space Administration Office of Naval Research	
6c. ADDRESS (City, State, and ZIP Code) 1308 W. Main ST. Urbana, IL 61801			7b. ADDRESS (City, State, and ZIP Code) NASA Langley Research Center, Hampton, VA 800 N. Quincy St. Arlington, VA 22217		
8a. NAME OF FUNDING/SPONSORING ORGANIZATION NASA & Organization Joint Services Electronics Program		8b. OFFICE SYMBOL (If applicable)		9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER NASA NAG 1-613 N00014-90-J-1270	
8c. ADDRESS (City, State, and ZIP Code) 800 N. Quincy St. Arlington, VA 22217			10. SOURCE OF FUNDING NUMBERS		
			PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.
			WORK UNIT ACCESSION NO.		
11. TITLE (Include Security Classification) Fault-Sensitivity and Wear-Out Analysis of VLSI Systems					
12. PERSONAL AUTHOR(S) Gwan Seung Choi					
13a. TYPE OF REPORT Technical		13b. TIME COVERED FROM _____ TO _____		14. DATE OF REPORT (Year, Month, Day) June 1995	
15. PAGE COUNT 148					
16. SUPPLEMENTARY NOTATION					
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)		
FIELD	GROUP	SUB-GROUP			
19. ABSTRACT (Continue on reverse if necessary and identify by block number) This thesis describes simulation approaches to conduct fault sensitivity and wear-out failure analysis of VLSI systems. A fault-injection approach to study transient impact in VLSI systems is developed. Through simulated fault injection at the device level and subsequent fault propagation at the gate, functional and software levels, it is possible to identify critical bottlenecks in dependability. Techniques to speed up the fault simulation and to perform statistical analysis of fault impact are developed. A wear-out simulation environment is also developed to closely mimic dynamic sequences of wear-out events in a device through time, to localize weak location/aspect of target chip and to allow generation of Time-to-Failure (TTF) distribution of a VLSI chip as whole. First, an accurate simulation of a target chip and its application code is performed to acquire real workload trace data on switch activity. Then, using this switch activity information, wear-out of the each component of the chip is simulated using Monte Carlo techniques. DTIC QUALITY INSPECTED 3					
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT <input checked="" type="checkbox"/> UNCLASSIFIED/UNLIMITED <input type="checkbox"/> SAME AS RPT. <input type="checkbox"/> DTIC USERS			21. ABSTRACT SECURITY CLASSIFICATION Unclassified		
22a. NAME OF RESPONSIBLE INDIVIDUAL			22b. TELEPHONE (Include Area Code)		22c. OFFICE SYMBOL

FAULT-SENSITIVITY AND WEAR-OUT ANALYSIS OF VLSI SYSTEMS

BY

GWAN SEUNG CHOI

B.S., University of Illinois, 1989

M.S., University of Illinois, 1990

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Electrical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1994

Urbana, Illinois

Accession For	
NTIS CRA&I	<input checked="checked" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution /	
Availability Codes	
Dist	Avail and/or Special
A-1	

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

THE GRADUATE COLLEGE

JULY 1994

WE HEREBY RECOMMEND THAT THE THESIS BY

GWAN SEUNG CHOI

ENTITLED FAULT-SENSITIVITY AND WEAR-OUT ANALYSIS

OF VLSI SYSTEMS

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

THE DEGREE OF DOCTOR OF PHILOSOPHY

Director of Thesis Research

N. Narayana Rao

Head of Department

Committee on Final Examination†

Chairperson

† Required for doctor's degree but not for master's.

© Copyright 1994 by Gwan Seung Choi

FAULT-SENSITIVITY AND WEAR-OUT ANALYSIS OF VLSI SYSTEMS

Gwan Seung Choi, Ph.D.
Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign, 1994
Ravishankar K. Iyer, Advisor

This thesis describes simulation approaches to conduct fault sensitivity and wear-out failure analysis of VLSI systems. A fault-injection approach to study transient impact in VLSI systems is developed. Through simulated fault injection at the device level and subsequent fault propagation at the gate, functional and software levels, it is possible to identify critical bottlenecks in dependability. Techniques to speed up the fault simulation and to perform statistical analysis of fault impact are developed. A wear-out simulation environment is also developed to closely mimic dynamic sequences of wear-out events in a device through time, to localize *weak location/aspect* of target chip and to allow generation of *Time-to-Failure* (TTF) distribution of a VLSI chip as whole. First, an accurate simulation of a target chip and its application code is performed to acquire real workload trace data on switch activity. Then, using this switch activity information, wear-out of the each component of the chip is simulated using Monte Carlo techniques.

DEDICATION

To my parents, TaeSun & YeongEui Choi

ACKNOWLEDGMENTS

First, I thank my advisor, Professor Ravi Iyer, for his scholarly expert guidance, persistent support, and patience. I thank Professors S. Kang, J. Patel, E. Rosenbaum, and D. Saab for serving on my Ph.D. dissertation committee. I would also like to thank my coworkers and friends at the Center for Reliable & High-Performance Computing for their friendship, especially Jim Barnett, Hungse Cha, Kumar Goswami, Alope Gupta, Wei-lun Kao, Sungho Kim, Inhwan Lee, Linda Lin, Devi Nair, Greg Ries, Jaidip Singh, Dong Tang, Timothy Tsai, Steven Vanderleest, and Fran Wagner. I am also grateful of Professor Fuchs for his support and guidance. I also thank Professor Saleh for providing insight into the SPLICE simulator. Most importantly, this work would not have been accomplished without the help and encouragement of my parents, Mr. & Mrs. Tae Sun Choi. Special thanks and gratitude go to my wife, Hyunjung, who patiently supported for completion of this work. Also, I thank my best friend and brother, SeungJin, and his wife for their support and encouragement. Lastly, I thank my daughter, Sarah, for bringing me great luck.

TABLE OF CONTENTS

CHAPTER	PAGE
1. INTRODUCTION	1
1.1. Related Research and Motivation	2
1.1.1. Transient fault	2
1.1.2. Permanent fault	7
1.2. Approaches and Impact	11
1.2.1. Mixed-mode simulation	11
1.2.2. Fault-dictionary approach	12
1.2.3. Coincidental fault analysis	13
1.2.4. Program upset analysis	14
1.2.5. Monte Carlo wear-out simulation	16
1.3. Thesis Organization	17
2. TARGET SYSTEMS	19
2.1. HS1602 16-bit Microprocessor	19
2.2. EEC131 Jet-Engine Controller	21
2.3. MC68000 Microprocessor	23
3. TRANSIENT FAULT IMPACT ANALYSIS	25
3.1. Mixed-Mode Simulation Approach	25

3.1.1. Simulation environment	28
3.1.2. Fault/error analysis	31
3.1.3. Statistical analysis environment	32
3.1.4. Experiment	35
3.1.5. Results (a). Transient fault severity (b). Pin error distribution (c). Charge Level Analysis	36
3.1.6. Empirical models (a). Error Propagation Model (b). State Transition Model	44
3.1.7. Discussion	51
3.2. Fault-Dictionary Based Approach	53
3.2.1. Circuit-level fault injection	54
3.2.2. Concurrent transient simulation	62
3.2.3. Illustration of the fault dictionary approach	64
3.2.4. Discussion	67
4. TRANSIENT FAULT MODEL	68
4.1. Validation of the Mixed-Mode Transient Simulation	70
4.2. Fault Model Comparison	72
4.2.1. High-level impact of transient fault models	74
4.2.2. Results	76
4.2.3. Discussion	78
5. COINCIDENTAL FAULT ANALYSIS	80
5.1. Coincidental Fault Injection Experiment	81

5.2. Impact of Single and Multiple Fault Injections	83
5.3. Multiple Fault Injection	85
5.4. Confidence Limits	86
5.5. Discussion	87
6. SOFTWARE UPSET ANALYSIS	89
6.1. The Experimental Approach	90
6.2. Fault/Error/Upset Analysis	92
6.3. Upset Classification	95
6.4. Fault Simulation	97
6.5. Results	99
6.5.1. Upset severity	99
6.5.2. Error propagation modeling	101
6.6. Discussion	104
7. PERMANENT-FAULT ANALYSIS	106
7.1. The Experimental Analysis Approach	107
7.1.1. The logic simulation	110
7.1.2. Monte Carlo simulation (a). Electromigration (b). Oxide breakdown	111
7.1.3. Importance sampling	124
7.2. Case Study	129
7.2.1. Experiment	129

7.2.2. Results	130
7.3. Discussion	135
8. CONCLUSIONS	137
8.1. Summaries	137
8.1.1. Transient fault sensitivity analysis	137
8.1.2. Permanent fault wear-out simulation	140
8.2. Future Extensions	140
REFERENCE	142
VITA	148

CHAPTER 1.

INTRODUCTION

Analysis of VLSI systems for reliability and fault sensitivity at the design stage is an essential step in avoiding the high cost of redesign and modification after the finalized design is submitted for fabrication. Automated fault/failure analysis techniques are needed to effectively evaluate alternatives in design tactics and to aid in the synthesis of architectures that meet required fault-tolerant specifications. A simulation approach can both evaluate the reliability of a chip and determine its fault tolerance capabilities.

This thesis describes simulation approaches to conduct fault sensitivity and wear-out failure analysis of VLSI systems. A simulation approach to study transient faults in VLSI systems is developed. Through simulated fault injection at the device level and subsequent fault propagation at the gate, functional and software levels, it is possible to identify critical bottlenecks in dependability.

A wear-out simulation environment is also developed to closely mimic dynamic sequences of wear-out events in a device to localize the *weak spot* of a target chip and to allow generation of *Time-to-Failure* (TTF) distribution of a VLSI chip as a whole. First, an accurate simulation of a target chip and its application code is performed to acquire trace data (real workload) on switch activity. Then, using this switch-activity information, the wear-out of each component in the chip is simulated using Monte Carlo techniques.

Section 1.1 presents the related research and motivates the simulation approaches. Section 1.2. presents the approaches and impact of this work, and Section 1.4 previews the organization of the rest of the thesis.

1.1. Related Research and Motivation

Research in the area of reliable and fault-tolerant design has been in progress for some time. A vast variety of methods for error detection, correction, and recovery have been proposed. Seldom, however, is the performance of these schemes evaluated in a realistic operating environment. To justify the cost for using a particular fault tolerance design implementation, it is necessary to examine the cost/benefit achieved by applying such a technique. This is especially true in VLSI systems, where the speed and area constraints can impose severe restrictions in designing for dependability. Although, in principle, such evaluations can be performed analytically, the many simplifying assumptions required tend to cast considerable doubt on the validity of the final design. The following two subsections introduce the related research in transient and permanent fault-analysis area, and motivate the necessity for the methodologies proposed in this thesis.

1.1.1. Transient fault

Traditional reliability analysis methods provide an evaluation of the impact of random failures in a system. Documented evidence has repeatedly shown that the majority of system failures are transient in nature, i.e., not due to broken parts. These transient faults can propagate and cause system failures by upsetting

program control flow or by propagating to the external environment via external I/O pins. Specifically, they can cause single or multiple upsets at the software level and have to be quantified to effectively evaluate alternative design strategies. Further, the resulting fault models can form the basis for the evaluation of an integrated system design in a hierarchical fashion. Hence, it is imperative that methods and automated tools to analyze the impact of transients on VLSI designs be developed.

An early study of failures in digital systems reported in [1] showed that nearly 90 percent of failures were transient in nature. Studies using failure data from IBM mainframes reported in [2] also showed that nearly 85 percent of major system errors were transient in nature. Naturally, this has led to valid concerns regarding the dependability of highly reliable systems. A particular source of concern is the impact of transients which are rather common in avionic environments.

Device Level: Device-level analysis of the mechanisms of transient upset has been in progress for quite some time. The hazards of transient upsets in dynamic RAMs were first reported in [3], where the behavior of alpha particle-induced soft errors was explored. Several simulation techniques for modeling the device-level effects of cosmic particle-induced transients have been developed (e.g., [4] and [5]). In [4], a SPICE circuit with a current source is used to represent the collected charges generated by alpha particles. An approximate analytic solution which models a current transient is proposed in [6]. The model includes parameters which represent the maximum current, the collection time

constant of the junction, and the time constant for initially establishing the ion track. The proposed analytic solution is validated by comparison with other computer models. In [5], a simulation technique for modeling the ion shunt effect is described.

Physical Injection: Experiments aimed at error analysis through the physical insertion of faults (via hardware or software) on the target hardware systems have been performed by several investigators. In [7], results of fault-injection experiments conducted on the *fault tolerant multiprocessor* (FTMP) are described. In the experiments, the pin-level logic is perturbed to study fault impact on the error detection and reconfiguration mechanisms. Experiments to study fault latency through hardware fault injections (at the NASA AIRLAB testbed) are described in [8] and [9]. More recently, in [10], a testbed facility to perform fault injections and to monitor the impact on a target system has been developed. The facility was used to validate a computerized interlocking system for the French railways.

Software Injection: Studies have also focused on software methods for analysis and detection of transient upsets. In [11], a simulation experiment to determine the efficiency of a number of error-detection mechanisms is reported. In [12], three major categories of upsets are studied: changes in data, temporary program divergence, and permanent jumps as a result of transients. Also, an experimental system to demonstrate the transition from a normal to an upset state is described. In [13], an approach to expose upsets in a computer system by abstraction verification is proposed. An assessment of different transient-error

test methods for microprocessors is presented in [14]. In [15], transient faults which result in steady-state failures are analyzed, and detection methods are presented. In [16], new experiments to study fault and error latencies using software techniques under varying workload conditions are discussed. Information gathered from these studies shows that the data generated can provide considerable insight into error manifestation.

Heavy Ion Radiation-Induced Injection: Several investigators have performed physical-fault insertions by radiating target chips with various ion sources. In [17], an analysis of the vulnerability of the Z80 microprocessor based on ion-bombardment testing is described. Upsets are associated with the machine cycle during which the errors first appear on the pins. Single event upsets in the MC6809E microprocessor due to heavy-ion radiation were reported in [18]. Several concurrent error-detection schemes for a microprocessor have been evaluated by physical-radiation induced fault injection in [19]. A novel method of inducing transients via heavy-ion radiation from a Ca^{252} source is also described in [19]. The method is applied to a MC6809E microprocessor. Recordings of the error behavior are used to characterize the errors, as well as to determine coverage and latency, for several error detection schemes. These studies show that the data generated can provide considerable insight into both error manifestation and fault impact.

The proposed hardware methods have several drawbacks. First, they offer no feedback to the system designers and are useful only as validation methods after the chip or system has already been built. Controlling the location and type

of fault injected is nearly impossible, and when faults are injected; monitoring and analyzing their impact on the chip's faulty behavior is extremely difficult.

Simulation-Based Fault Injection: An important question not addressed in the studies described is the propagation of transients from the device level through the microprocessor functional units and pins. In addition to furthering the knowledge of transient-fault propagation in microprocessors, this information is crucial for defining the vulnerability of microprocessors to transients. Simulation of fault propagation in an avionic microprocessor was conducted in [20]. In [21] and [22], preliminary experiments to quantify the impact of transients from the device to the pin level were described. Transients with low charge levels (0.5 pC - 4.0 pC) were injected. The ensuing logic upsets and first-order latch, and pin errors were analyzed via analysis of variance methods. Recently, in [23], a simulation model of the IBM PC was developed and injected with gate-level transient faults. A general fault-prediction model based on instruction execution was developed, along with a model of the resulting fault manifestations. The detailed description of several important simulated-fault injection studies are reported in [24]. Table 1 summarizes features and representative studies in simulated-fault injections at different levels.

An important question that must be addressed is the propagation of transients from the device level through microprocessor functional units and pins. This information is crucial for defining the vulnerability of microprocessors to transients. Some of the key issues are: modeling of device-level transient faults, simulation of error manifestation on a chip-wide scale, injection of a large

Table 1: Simulated-fault injection studies.

Category	Electrical Level	Logic Level	Function Level
Approach	Alter electrical current and voltage in circuits	Inject stuck-at or inverted faults to logic gates	Inject faults to CPU memory, I/O devices, etc.
Target	VLSI chip	VLSI chip	Computer system
Under Study	Software running on the chip	Computer system Software	Network system Software
Studies	Fault simulation[25]	BDX[26]	Trace-driven[16]
	HS1602[21]	BDX930[20]	NEST[27]
	FOCUS[22]	IBM RT PC[23]	DEPEND[28]
			REACT[29]

number of faults to obtain statistically valid results, execution of test-software sufficiently large to exercise a significant number of the circuit functions, and monitoring of the fault impact.

1.1.2. Permanent fault

Among all of the IC technological trends, scaling has always been an important method for reducing die size and thus increasing circuit performance and complexity. Scaling of design layout rules can lead to increased electrical stresses, and this in turn can accelerate the wear-out process. In this context, I attempt to address wear-out-related permanent failures in VLSI systems in this thesis.

In [30] the impact of device-dimension scaling on wear-out mechanisms is discussed. Several methods have been proposed in the literature which discusses specific wear-out/failure mechanisms.

Electromigration: An early investigation [31] of the electromigration process shows that the *mean time-to-failure* (MTTF) of a conductor under a constant current stress can be expressed by an empirical equation. The proposed (Black's) equation is widely accepted and is used to model reliability of IC devices. The equation is experimentally verified in [32] and [33]. Based on the model mentioned, a number of approaches to analyze and predict the reliability of ICs due to electromigration failure mechanism have been proposed [34], [32] and [35]. In [34], Φ (average current density used in Black's equation) is obtained via circuit simulation, and Black's equation is used to predict device reliability. Chip reliability is calculated assuming independent failure of nodes. The operational MTTF for electromigration failures by extrapolating the results obtained via accelerated testing is predicted [35].

All previous semiempirical methods rely on the generation of device reliability to obtain MTTF for the system. It is possible to generate the system failure distribution using the MTTF of each device. But this assumes that node failures are independent, which may not be true in practice. Such methods cannot accurately predict the variance and other characteristics of the distributions of a reliability measure (e.g., TTF).

In [36] a physical experimental study to determine the device TTF due to electromigration is described. The test circuits consisting of metal lines of varying lengths and widths were fabricated and tested under different current stresses. A log-linear relationship was found between the TTF and operating voltage (or fabrication-scale). In [37], an experiment to determine the MTTF due to

electromigration with pulsed, rectangular current applied at the metal line is described.

A method using Monte Carlo simulation for electromigration is described in [38]. This approach starts from basic physical principles, such as the continuity equation and diffusion laws, to drive simulation models built to mimic the dynamic sequence of events in a modeled target device using Monte Carlo method. In general, these techniques require a great amount of time and are not easy to apply at the VLSI level.

Dielectric breakdown: That the time-dependent dielectric breakdown has been one of the major failure mechanisms for MOS IC is pointed out in [39] and [24]. A number of models to depict the breakdown mechanism of dielectrics are proposed in [40] and [41]. Among the proposed mechanisms, charge-trapping model in the oxide is commonly accepted as the major failure mode [40]. The empirical equation used to describe this phenomenon is given in [32]. In [40], a more elaborate empirical equation is derived to recognize the temperature dependence of T_{bd} . Several reported experimental results with the theoretical model are compared in [39].

There is considerable evidence to show that the failure rate of a system is a dynamic function of the system activity. Statistical evidence shows that there is an increased probability of failure of logic devices at higher activity levels [42], [21] and [43]. To accurately predict the reliability of a complex circuit, switching behavior of each component and the subsequent effects must be comprehen-

sively modeled and analyzed. A methodology to locate areas or design features that are susceptible to common wear-out/failure mechanisms must be developed. In particular, the wear-out mechanisms involved in each device failure and their effects at the system level are investigated and modeled. Apart from furthering the knowledge of wear-out processes in microprocessors, this information is crucial for further defining the vulnerability of microprocessors to common wear-out mechanisms.

In our approach, accurate simulation of the target chip and its application code, using a mixed-mode hierarchical simulator, SPLICE, is performed to acquire trace data (real workload) on switch activity. Then, using this switch activity information, wear-out of the entire chip is simulated using Monte Carlo techniques. The wear-out mechanisms involved in each device failure and their effects at the system level are quantified. Also, areas and design features that are susceptible to common wear-out/failure mechanisms are located.

1.2. Approaches and Impact

1.2.1. Mixed-mode simulation

An effective way to evaluate a HW/SW design for reliability is to observe the actual behavior of the design under faulty conditions. During the design phase, however, this evaluation is possible only through a simulation approach. To accurately model the device-level faults and to simulate their manifestation at the chip-wide scale, a mixed-mode simulation approach must be taken. A target system must be simulated for a reasonable length under a realistic operating condition, while device-level faults are injected in electrical-level analysis, such as SPICE, and subsequent logic-level impact are propagated at the gate and higher levels.

This thesis contains a hierarchical mixed-mode simulation approach for the runtime injection of transients and for the tracing of their impact. A determination of the probability that a transient results in latch, pin or functional errors can be made. The approach also allows quantification of the impact of transient hardware errors at the software execution level.

Faults are injected at runtime, at the transistor level, and their propagation to the I/O pins is monitored and analyzed. The type of functional errors which can result from transients is determined. To isolate the critical paths in the circuit, the fault propagation between the functional units and the external pins is quantified. In particular, the mechanisms involved in internal propagation of latch errors and their effects at the pin-level are investigated.

The approach is illustrated with a case study of the impact of transient errors on a microprocessor used in jet engine. The chip's vulnerability to transients is quantified. For each functional unit, the transient-induced error distribution at the external pins is determined. A state transition model is constructed to describe the error propagation between the microprocessor functional units and the subsequent distributions at the I/O pins. The model is used to identify and isolate the critical fault propagation paths, the functional units which are most sensitive to fault propagation and having the highest potential of causing external pin errors.

An empirical model to depict the process of error explosion and degeneration in the target system is derived. The model shows that, if no latch errors occur within eight clock cycles, no significant damage is probable. Thus, the overall impact of a transient is well contained. A state transition model is derived from the measured data to describe the error-propagation characteristics within the chip and to quantify the impact of transients on the chip's external environment. The model is used to identify and isolate the critical fault-propagation paths, the module most sensitive to fault propagation and the module with the highest potential of causing external pin errors.

1.2.2. Fault-dictionary approach

Mixed-mode simulation requires a large amount of both processing power and storage space. We must be able to perform near-exhaustive fault simulations necessary to obtain a valid fault sensitivity analysis. Thus, we developed a fault-injection methodology, based on a look-up table (fault dictionary) that

contains device-level fault-behavior patterns, for a fast logic-level fault simulator.

The fault-dictionary approach is a method to reduce the time requirement for simulation of a massive number of device-level transient faults. A device-level fault-behavior dictionary, for logic-level fault injections, is generated from extensive circuit-level fault simulation with SPICE. Fault injection locations and the gates around those locations are extracted and evaluated with SPICE. The extracted subcircuits are exercised with exhaustive combinations of inputs while fault injections are performed. Faulty behavior at the outputs for each subcircuit is recorded in the dictionary according to the subcircuit's input vector, fault-injection time, and location. The generated dictionary can be used to inject, in runtime, the logical error pattern on the target design. Concurrent simulation can be performed using the fault-behavior dictionary approach to allow simultaneous evaluation of a great number of faults in single simulation pass. The fault-dictionary approach is illustrated by a case study of a MC68000 microprocessor.

1.2.3. Coincidental fault analysis

An approach intended to investigate critical aspects of such designs from a fault-tolerance viewpoint is also developed. The method is illustrated using an example of a fault-tolerant jet engine controller. In particular, the digital aspects of the dual-channel controller, described at the logic and functional levels, are simulated, and transient fault injections are performed. The coverage of the dual technique to tolerate single and multiple transients is evaluated.

In the simulated controller, fault detection and reconfiguration are performed through transactions over communication links. Instructions specifically designed to exercise this cross-channel communication are executed. The simulated fault-injection approach is illustrated by measuring the level of effectiveness of the dual configuration to transient errors. The results show that none of the single injections affects more than one channel, while approximately 12 percent of the multiple injections affects both channels.

1.2.4. Program upset analysis

Another important question that must be addressed is the propagation of faults from the device level through the microprocessor functions and the subsequent impact on the application software that is executing on the chip. No study that combines hardware and software fault behavior analysis has been reported. Also, modeling of software upset is necessary for high-level design evaluation. Faulty behavior of the software under a fault condition should be quantified and modeled.

A transient fault can be activated and propagated to corrupt logic values in latches in its propagation path. Once a fault propagates to a latch (i.e., latch error) it can either relatch, propagate out to the external I/O pins and/or disappear in each clock cycle. Once an error is present in the hardware, there is a chance that values at data or address registers/buses can become corrupted.

Software(program-flow level) upset could occur when one or more of those values are corrupted, e.g., the program could jump to a wrong address or the program result could be incorrect.

A systematic approach to quantifying the program-flow-level upset sensitivity of VLSI designs is required. The types of upsets which can result from fault injections must be determined because an error in the program counter register can cause a program-flow deviation. The mechanism involved in error propagation within a chip and the effect at the program-flow level has to be investigated. Also, the analysis shows that there is a significant chance of multiple software upsets occurring. Multiple program upset occurs since the erroneous values can propagate on fanout paths to multiple locations on the chip. The results suggest that present methods of validation that assume single upsets may be inadequate.

This thesis contains methodologies for integrating and exporting the quantifiable fault/error-sensitivity measures to the system-level analysis. In the proposed experimental approach, determination of the error characteristics at the software level can be performed. The real-time characterization of these measures can be incorporated into system-level analysis where other higher-level dependability analysis can be performed using tools like DEPEND [28]. The impact of faults on system functionality, particularly the quantification of faulty behavior at the program-flow level, is a central consideration.

1.2.5. Monte Carlo wear-out simulation

To effectively evaluate a long-term reliability, a method for predicting permanent faults in VLSI designs is required. An approach that combines the switch-level circuit simulation and the device-level Monte Carlo simulation to achieve realistic reliability assessment is presented in this thesis. A target system is first simulated at the switch level, and trace data on switching activity is collected. This trace data is then used along with Monte Carlo simulation to model wear out at the device level, due to different failure mechanisms. The proposed approach currently supports two failure mechanisms: electromigration and gate oxide breakdown. The Monte Carlo analysis uses *importance sampling* to reduce the run lengths. The key advantage of this approach is that it can closely mimic dynamic sequences of events in a device through time for the specified failure mechanism(s). The technique can localize weak location/aspect of target chip, and can generate the TTF distribution of a VLSI chip as a whole, based on traces from circuit simulation using actual application codes and under realistic operating conditions.

The use of this approach for evaluating the reliability of a design is illustrated with a case study of the HS1602 microprocessor. The simulation analysis is performed under varying operating environments and fabrication-technology parameters. In particular, operating voltage, temperature and the device dimension are varied, and the reliability impact of reduced dimension and technology improvements are quantified. The method is illustrated by using it to predict the TTF characteristics of a microprocessor chip in a typical operating environment.

1.3. Thesis Organization

This thesis is organized as follows. Chapter 2 describes the target systems used to illustrate the methodologies developed to study fault propagation, fault sensitivity analysis and wear-out failure analysis. Chapter 3 presents the transient-fault impact analysis. The mixed-mode approach to performing various fault-sensitivity analysis is described, and the fault-dictionary-based approach to performing ultrafast transient-fault simulation is given. The target systems described in Chapter 2 are used to illustrate various proposed analyses. Chapter 4 contains the validation of the mixed-mode simulation model for transients and the comparison study to show that there is a statistically significant difference between the accurate device-level model and the often-assumed abstract logic-level fault model. Chapter 5 presents a correlated fault-injection experiment aimed at studying the fault-tolerant features of the target system. Chapter 6 describes software upset analysis. The propagation of faults from the device level through the microprocessor functions and the subsequent impact on the application software that is executing on the chip is studied. Chapter 7 presents the fault-dictionary approach that allows fault-sensitivity analysis of VLSI designs via device-level current-transient injection, logic-level fault propagation and monitoring of the fault impact. The approach combines electrical-level circuit simulation to generate device-level fault-dictionary and gate-level concurrent fault simulation to simulate a large number of faults rapidly. Chapter 8 presents a methodology to locate areas or design features that are susceptible to common wear-out/failure mechanisms. In particular, the wear-out mechanisms involved in

each device failure and their effects at the system level have to be investigated and modeled. The last chapter summarizes the work presented in this thesis and gives directions for future research in the area.

CHAPTER 2.

TARGET SYSTEMS

2.1. HS1602 16-bit Microprocessor

The first of the three target systems for our study is a microprocessor used for real-time control of jet-engine functions. The system is presently used in commercial aircraft, including the BOEING-747 and the 757. The controller (EEC131, manufactured by Hamilton Standard), which is the next target system, has two channels; the processing elements of both channels are identical. The system has a real-time reconfiguration mechanism wherein the lead channel stops its usual operation on detecting a fault and transfers control to the dual channel. The system incorporates a variety of fault-tolerant design features at different levels including software checks, parity checks, memory test and error counting.

The control system samples engine parameters such as the fuel flow, the temperature, the engine speed and other external inputs such as air speed and positional parameters. The sampled parameters are digitized and updated into the RAM approximately every millisecond for further processing. The controller also reads pilot inputs (from the throttle and various switches) into a RAM work area and calculates the desired control functions. The calculated functions are used to drive display indicators and to control the engine. The equations describing the control functions are programmed in the application code which resides in EPROMs.

The control system architecture thus contains microprocessors, memory units, I/O gate array chips, communication channels, frequency samplers, A/D converters and D/A converters. In this experiment, the microprocessor and its associated memory were simulated with a focus on the impact of transient errors.

The 16-bit HS1602 microprocessor (Figure 1), which is the heart of the controller, consists of six major functional units. The *arithmetic logic unit* (ALU), which contains six registers, can perform double precision arithmetic operations. The control unit, which is responsible for issuing signals to control the operations of the ALU, is made up of combinational logic and several registers. The decoder unit decodes I/O signals, the multiplexor unit provides the discrete lines

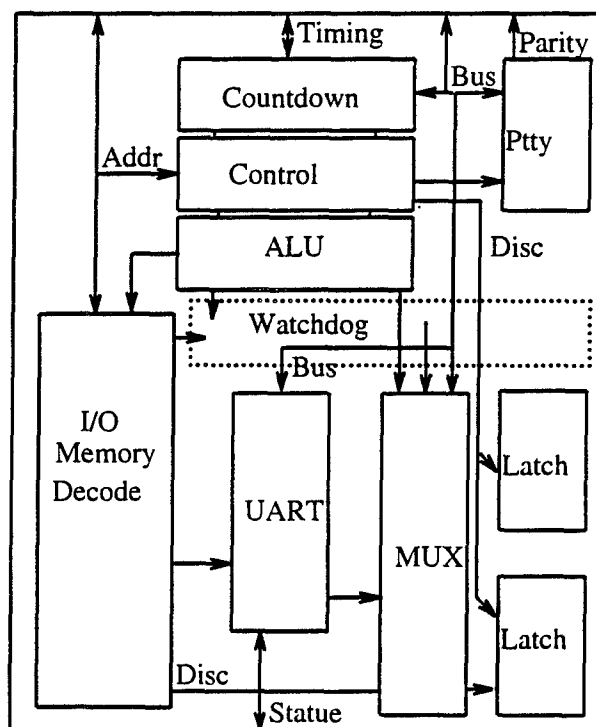


Figure 1: Data flow diagram of HS1602.

and buses, and the countdown unit is used to drive chip-wide clock signals. The watchdog unit provides protection against faults by resetting the processor in the event of a parity error or in the event when the application software is timed out by the software sanity timer. The signal to synchronize the dual system is also provided by this unit. The chip runs at 6 MHz and is implemented in a 3- μ technology CMOS gate array made of 2688 blocks of 4 N-channel and 4 P-channel transistors.

2.2. EEC131 Jet-Engine Controller

The second target system in our study is a microprocessor-based, dual-channel controller for real-time control of jet-engine functions. The fault tolerance achieved by the redundant design is tested in the Section 6. The system processes data obtained from dedicated engine sensors to provide several functions, such as automatic thrust control, engine-limit protection, engine-transient control, engine fuel and oil temperature management and thrust reverser control. The digital system architecture (Figure 2) contains microprocessors, buses, memory units, I/O processors, asynchronous serial communication links, frequency samplers and A/D converters.

The controller has two independent channels, referred to as channel A and channel B, each consisting of a microprocessor chip. The I/O configuration of the hardware is identical for both channels, with the exception of the following three control loops: the turbine-cooling air loop and a thermatic rotor-control loop assigned only to channel A, and another thermatic rotor-control loop

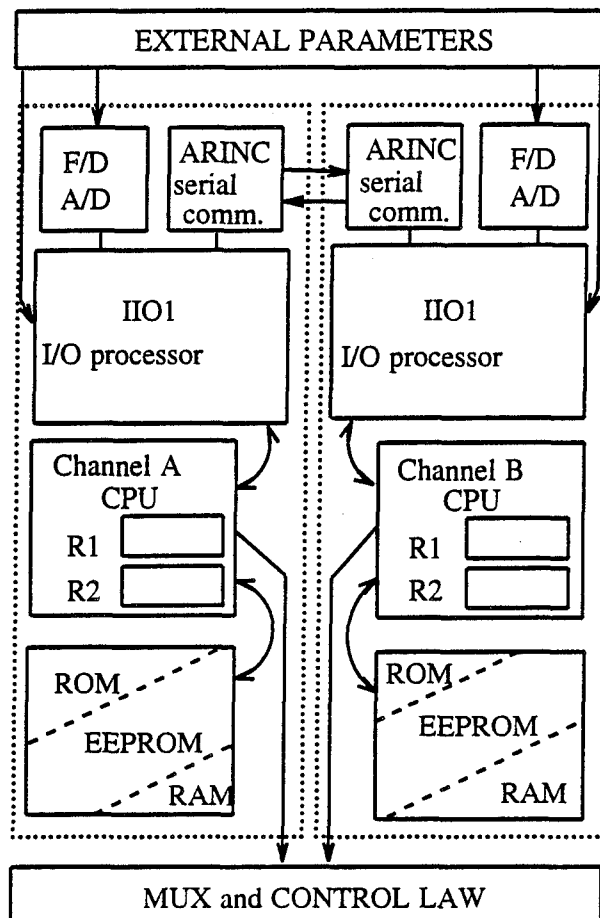


Figure 2: EEC131 jet-engine controller.

assigned only to channel B. All other functional loops have redundant implementations and can be controlled by either channel. In each channel, input information (e.g., temperature, test and drift signals, resolver inputs, transducer inputs, torque motor wraparounds, rotor speeds, pressures and test signals) is first digitized by two on-board frequency samplers and an A/D converter. This digitized input data (a single word) is then stored in the channel CPU register R1. The digitized data in channel A is sent to channel B via a serial communication link and stored in CPU register R2 of channel B. Similarly, the data in R1 in channel

B is sent to register R2 in channel A. In each channel, a logic comparison of the contents of registers R1 and R2 is made. If the comparison fails, a *range test*, which compares the data in R1 and R2 with the range information recorded in the ROM for the particular input variable, is performed. The content of the register within range is then used for continued processing. If both the data in R1 and R2 are out of range (multiple failure), the control signal (output) is synthesized from the other parameters and from previous engine states recorded in the EEROM. In cases of sustained multiple failures, a background test routine identifies the failed channel and transfers the engine control to the working channel.

The ability to detect faults and reconfigure a system through comparisons and range tests of critical input data is the main fault-tolerant aspect of the controller. In our experiment, a single-channel functional error is assumed to occur if the injected transient alters the contents of either CPU registers R1 or R2 in a single channel. A dual-channel functional error is defined as an event in which the contents of CPU registers R1 and R2 are faulty in both channels. This event would require the invocation of the next level of protection.

2.3. MC68000 Microprocessor

The third target system analyzed in our study is Motorola's MC68000 general-purpose microprocessor. The MC68000 architecture is comprised of a 16-bit data bus and 24-bit address bus. It can directly access 16 megabytes of memory and has 16 32-bit general purpose registers, a 32-bit program counter,

and an 8-bit condition code register. The first eight registers are used as data registers for byte(8-bit), word(16-bit), and long word(32-bit) operations. The second set of seven registers and the stack pointer may be used as software stack pointers and base address registers. In addition, the address registers may be used for word and long word operations. All of the 16 registers may be used as index registers. The details of the microprocessor are described in [44].

CHAPTER 3.

TRANSIENT FAULT IMPACT ANALYSIS

3.1. Mixed-Mode Simulation Approach

The mixed-mode simulation approach allows fault-sensitivity analysis of VLSI designs through simulation. A mixed-mode hierarchical simulation of VLSI systems can be performed, with runtime-fault injection, for a range of user-specified faults. Fault injections occur at the device level, and fault propagation is studied at the gate and higher level. Figure 3 depicts the overall experimental approach. The approach takes as input a netlist of the hardware description of the system and converts it into a simulation model¹. SPLICE is used as the simulation engine.

The *fault injection* process is implemented as a runtime modification of the circuit, whereby a current source is added to a target node,² which alters the voltage level of the node over the time interval of the injected-current waveform. The experimental approach allows both transient and permanent (single or multiple) fault injections, although the thrust of the present work is on transients. Since the injected-current source is specified as a mathematical function, the resulting transients can be of varying shapes and durations. For example, electrical power surge, in-chip alpha-particle intervention, lightning, and bridging faults

¹Hardware description (netlist) in various formats(e.g., Tektronics, Hamilton Standard) can be translated into the BLT simulation model description which is recognized by SPLICE.

²A node is defined as a point in a conductive interconnection between electrical and/or logical elements.

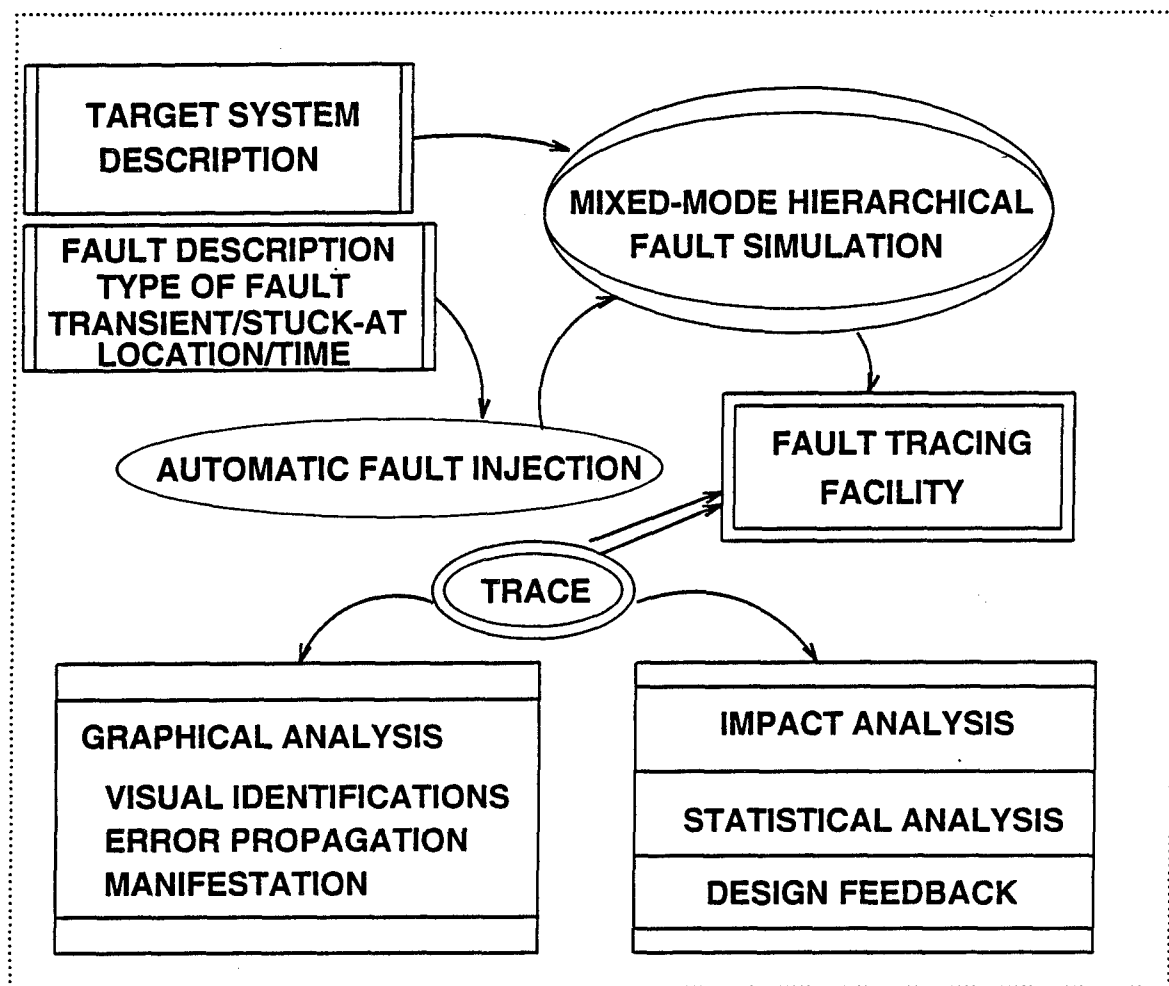


Figure 3: The mixed-mode simulation approach.

can be modeled. The user can control the location of a fault, the time and duration of a fault, and the shape of the current source.

The *tracing facility* monitors all switching activities in the target system, including fault propagation through each gate or transistor, for all processed events. The trace data for each event consists of the time of the event, the hierarchical node name, and the new and previous voltage levels (for electrical nodes), or the new and previous logic levels and their strengths (for logic nodes).

The *graphical analysis facility* is used to visualize the error activity in different functional units of the processor and the fault propagation on the major interconnects and at the external pins. In the usual case, the preprocessed error data from the fault simulations form the input to the graphical program, which allows accelerated viewing of the impact of the injected transient. The propagation path of an injected transient is traced on screen by a red *blip* through various internal modules and external pins. The fault-sensitive functional unit (originally represented in blue) gradually becomes yellow, then red. After each injection/simulation run, the statistical distributions of the latch and pin error characteristics and the fault propagation are calculated and displayed.

Quite apart from the obvious advantage of visualization, the specific design advantage of this approach is that it allows fast identification of vulnerable units even prior to obtaining a detailed quantification. For example, in the target system discussed in the following section, the graphical results show that the *watch-dog* unit, a critical component, was a major source of error propagation, even though very few errors did indeed affect the unit itself. Without the display, such a result would require considerable analysis. The details of the graphical analysis facility are described in [22].

The *statistical analysis tools* provide impact analysis of the target system and generate the models necessary to depict the fault behavior in the system (e.g., I/O pin error distribution, latch error distribution, and internal fault-propagation model).

3.1.1. Simulation environment

The electrical analysis in SPLICE1 [45] is based on the method of *Iterated Timing Analysis*(ITA). This technique incorporates a nonlinear relaxation method together with event-driven selective tracing. ITA has been shown to be as accurate as SPICE2 (assuming identical device models) and can provide a speedup of up to two orders of magnitude. The logic analysis in SPLICE1 is performed by using a relaxation-based method that uses MOS-oriented models. Virtually unlimited levels of signal strength can be associated with each of the logic values in order to further enhance accuracy. This approach allows a correspondence between the electrical output conductance and the logic output strength. By using a fanout-dependent delay model, which is capable of handling first-order effects, accurate delay handling is achieved.

Figure 4 shows how fault injection was incorporated into the mixed-mode simulation. Initially, the circuit-model description is read and modified by adding a current source at the fault injection node. During each iteration, the scheduled node events in the current time step (virtual simulation time) are processed, and new events are scheduled and queued in the event list. For each fanin element in the processed node, the element type (electrical, switch-level, logical) is determined.

If the element type is electrical, then additional analysis (see *Electrical Analysis* in Figure 4) is performed to determine if the node is an injection site (i.e., analog signals are dealt with). If the node is an injection site and if the

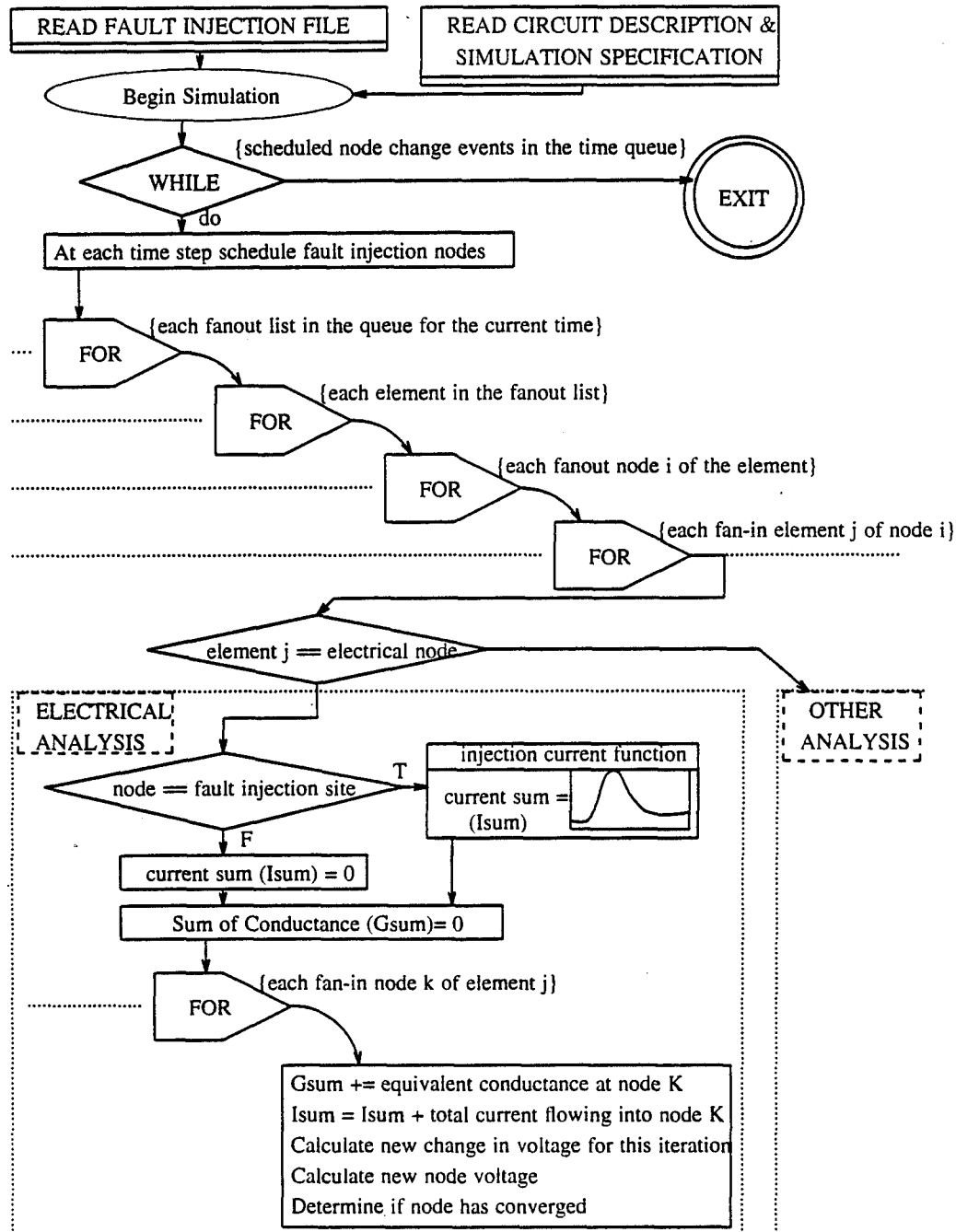


Figure 4: Fault injection algorithm.

virtual-simulation time is within the fault-injection time window (the time period between fault-injection time t and $t + dt$, where dt is duration of the fault), the current source representing the transient is activated, and additional current is

added to the total current calculation. The additional current value is determined from a function representing the current source for the particular virtual-simulation time. The total current is used to calculate *fault* voltage level at the processed node. The actual design parameters of a VLSI circuit and the capacitances extracted from the circuit layout can be used in the electrical-level analysis. The specific waveforms used in the transient-fault simulations follow the double-exponential function proposed in [6].

$$I(t) = \zeta [e^{-t/\alpha} - e^{-t/\beta}]$$

where ζ is the approximate maximum current, α is the collection time constant for the junction and β is the ion track establishment time constant. This function depicts the current-transient response for an ion-particle penetration of a diffusion area.

In the simulations, the gates around the region of the fault injection are simulated at the electrical level, and the rest of the processor is simulated at the logical level, as shown in Figure 5. A simple CMOS AND gate with buffered output is illustrated in the figure. The dotted boxes indicate normal voltage waveforms for the circuit, and the dashed boxes contain waveforms resulting from a transient injection at the location marked by X. Note that waveforms within the electrical-level analysis behave in an analog fashion but are discrete in the logic-level analysis.

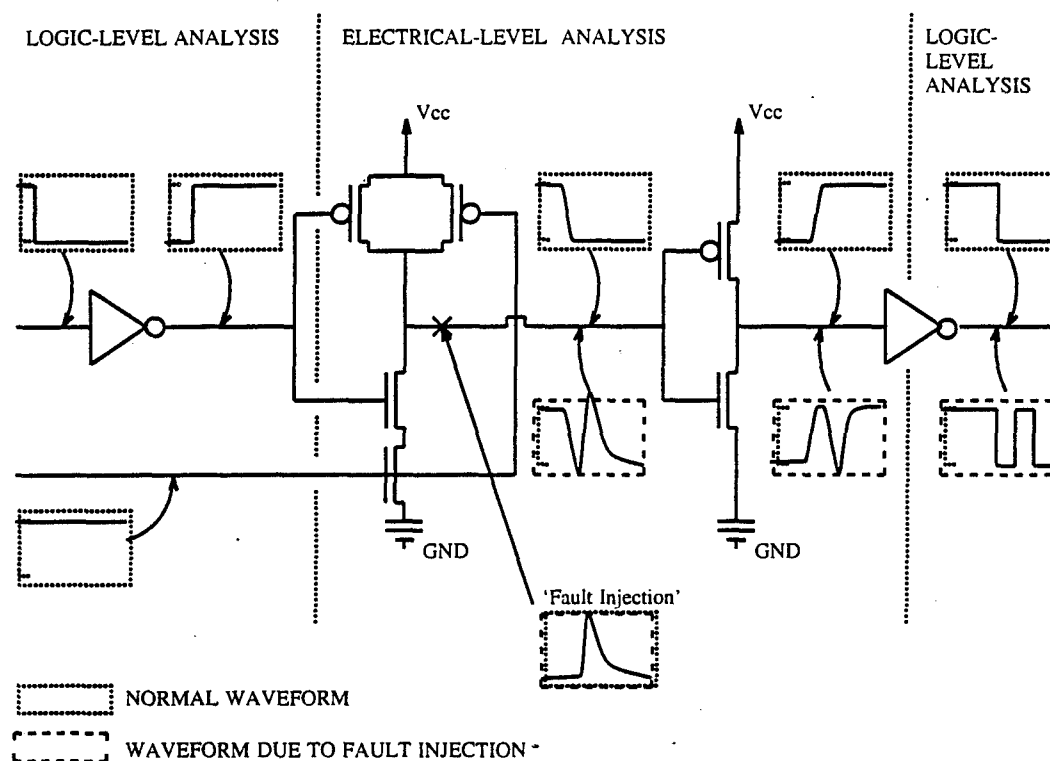


Figure 5: Fault injection - electrical to logic level.

3.1.2. Fault/error analysis

Recall that the tracing facility is capable of monitoring each node. The error data for the analysis are generated by comparing each faulted simulation with a fault-free simulation. An error is assumed to occur if the injected transient caused the node voltage to vary beyond a defined-logic threshold. For each simulation, the recorded data include the time of fault occurrence, the location of fault, the faulted value, and the fault-free value. Each fault event is also classified as either a timing error (premature or late firing) or a value error.

Statistics are collected on the fault injections that result in a voltage transient large enough to produce latch and pin errors and errors at the interconnections of

the functional units. A latch or pin error is assumed to occur when an erroneous value is detected at the output of a latch or pin. Statistics on errors altering the processor functions are also collected. The collected statistics are classified by the charge level and by the location.

3.1.3. Statistical analysis environment

The statistical analysis environment quantifies the fault sensitivity of the chip by evaluating a number of measures. Analysis of the error data is performed to determine the effects of different charge levels on the injected transients, and on the severity of latch, pin and functional errors. Models to depict error propagation both within the chip and to the external environment are derived. The following terminology is used in the analysis.

- 1.) Latch errors: fault injections which result in voltage transients that cause errors in latch outputs.
- 2.) Pin errors: fault injections which result in voltage transients that cause errors at the chips I/O pins.
- 3.) Functional errors: fault injections which result in altering the output control functions of the processor, e.g., an error resulting in a faulty write-enable signal.

- 4.) First order error: an error which occurs during the first clock cycle following a transient fault.
- 5.) Second and higher order errors: errors that occur during the second and subsequent clock cycles.³

Figure 6 summarizes the analysis performed. The statistical analysis provides measures of the severity of the errors resulting from the injected transient. The probabilities of latch, pin and functional errors are calculated. External pin-error distributions resulting from transients in the different functional units are obtained. Charge threshold for the different error types are also determined.

The data are used to construct two types of fault-propagation models. First, an overall view of fault propagation is provided by generating an empirical model to depict the process of error explosion and degeneration within the chip and to the external environment. Next, a state-transition model to quantify the probabilities of internal module-to-module latch-error propagation and subsequent propagation to the external environment is derived. A latch error can either be relatched, can propagate out to the I/O pins and/or can disappear during each clock cycle. Error explosion occurs if the erroneous values propagate on fanout paths to multiple locations on the chip. Fault propagation usually occurs because errors can get latched and then migrate to different sections of the chip. A latch

³ Transients modeled in the experiment last no longer than one clock cycle. Thus, no latch-error can occur directly from an injection after the first clock cycle. This is typical of effects of cosmic-rays and the like.

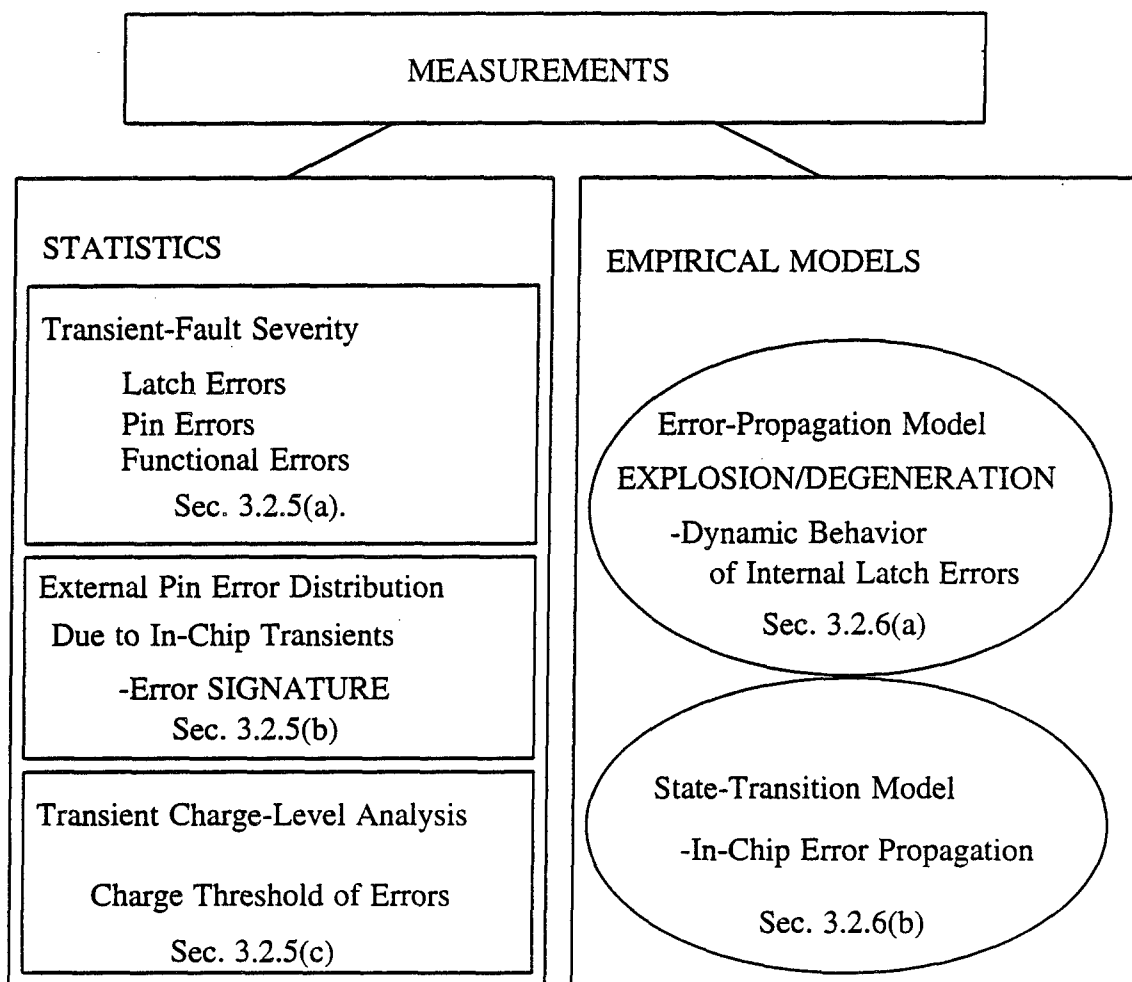


Figure 6: The analysis environment

error can stay undetected until it migrates to the pins at a later time. The additional internal propagation between latches can increase the probability of generating functional errors. Thus a characterization of the latch-to-latch fault-propagation patterns is important. The detailed methodology of the analysis is illustrated by the case study in the next section.

3.1.4. Experiment

The HS1602 microprocessor was used to demonstrate the mixed-simulation approach. Nearly 80 instruction cycles (90300 nsec) of the application code were executed on the target system during each simulation period. Simulation resolution was 1 nsec or less. The suite of application codes was carefully selected to ensure that all of the functional units were exercised. Each fault-injection simulation run took on the average two hours on a SUN 3/50 workstation. A total of 2100 simulations (lasting about three weeks) were performed. The figure 2100 represents the number of simulations required to obtain stable results. Increasing the number of fault injections beyond this value did not significantly change the nature of our results. During the simulation all the nodes (including all latches and external I/O pins) in the circuit were monitored and processed.

To establish simulation accuracy, a fault-free simulation run was compared with the operation of the actual hardware unit. Two comparisons were made. The first verifies the correct flow and execution of instructions by monitoring data, address, and control lines during the times that these lines are stable(logically valid). Correct execution flow was observed for the entire simulation period. The second comparison, which is more rigorous, monitors all changes in logical values, including transition times. This comparison reveals the electrical characteristics of the simulation, such as propagation delay, race conditions and gate loading. The result showed that the change in the logical values for both the hardware unit and the simulated model are identical, but the times of the transitions are slightly skewed. The timing skew was due to the variance of

several design parameters. Gate delay, for example, is specified as minimum, typical, and maximum delay time. A normalization was needed, because the actual system runs at 12.08 MHz and the simulation was set at 82 nsec per clock cycle (12.195122 MHz).

3.1.5. Results

This subsection provides a detailed illustration of the analysis methodology using the actual results for the target system. The impact of transients at the latch, pin and functional levels are determined. Also, the charge-level impact and I/O pin-error distribution are investigated, and empirical models of fault propagation are derived from the data. These results provide an overview of the fault propagation within the chip, and as such give a general evaluation of the fault sensitivity of the design.

3.1.5(a). Transient fault severity

Table 2 summarizes the overall impact of transients in the range 0.5 to 9.0 pC.⁴ In the table, a first-order error is defined as one which occurs during the first clock cycle following a transient-fault injection; second- and higher-order errors are those occurring during the second and subsequent clock cycles.⁵ The second column shows the number of fault injections that result in errors. The third

⁴The charge levels chosen represent transient response of various heavy ions including 100 MeV Fe ions, which are commonly found in the cosmic environment [46]. These levels were chosen to ensure that no permanent errors occur. Charge levels approximately greater than 10 pC are known to cause permanent latch ups (device failure)[47].

⁵Transients modeled in the experiment last no longer than one clock cycle. Thus, no latch error can occur from direct propagation after the first clock cycle. This is typical of effects of cosmic rays and the like.

column shows the total number of resultant errors, and the fourth column shows the 90 percent confidence interval.⁶ For example, out of 2100 fault injections, one or more first-order latch errors occurred in 470 cases (22.4 percent), and a total of 2149 latch errors were observed.

Table 2: Transient fault severity.

Type	Occurrences	Total Error #	Percentage	90% C.L.
Injected Transients	2100		100%	
First-Order Latch Errors	470	2149	22.4%	21.93%-22.82%
Second-and Higher-Order Latch Errors	120	1829	5.7%	
First-Order Pin Errors	255	1168	12.1%	12.09%-12.19%
Second-and Higher-Order Pin Errors	90	839	4.3%	
Functional Errors	193	747	9.2%	9.16%-9.22%

A number of issues relating to the fault sensitivity of the chip are highlighted by data. First, they show that over 20 percent of the injections result in latch errors. Given that a transient results in a latch error, the chance of multiple errors is high (an average of four latch errors per transient). The existence of such multiple latch errors is potentially a serious problem, since these errors can subsequently propagate to the pins and lead to multiple failures.

In addition, even though only 25 percent (120 out of 470) of the latch errors propagated past the first clock cycle (i.e., the first order), each such propagation can result, on the average, in about 15 latch errors, thus further intensifying the

⁶ The probabilities of failure of each independent trial(fault injection) is p , and thus random variable X (error occurrences) has binomial probability distribution with $n = 2100$, $p = p$ and $q = 1 - p$. \bar{X} for first-order latch error in our experiment is 470. Thus estimation of \bar{p} is 0.224 (= [number of first-order latch errors]/[number of fault injections]). Since n is sufficiently large ($n = 2100$), the probability density function of X can be approximated with a normal distribution with ($\mu = np = 470.00$, $\sigma^2 = np(1-p) = 364.72$). We estimated 90 percent confidence limits of p by normal distribution.

propagation problem. An effect of second- and higher-order latch errors is an increase in the probability of functional errors (erroneous control signals or data which result in an alteration of the microprocessor functions).

There is almost a ten percent chance of having functional errors. Over one-third of the total number of functional errors were due to transients in the ALU. Further analysis of the error data showed that a significant number of functional errors due to transients in the ALU were due to first-order effect because transients that latch directly on the ALU registers result in an immediate alteration of address or data information. Functional errors caused by second- and higher-order effects of transients were more dispersed among different functional units. A relationship between the second- and higher-order latch errors and functional errors is discussed further in subsection 3.1.5(c).

Table 2 shows that the percentage of first-order pin error occurrences is significant (over ten percent). Given a pin error, the chance of recurrence during the subsequent clock cycles is relatively high (90 out of 255) and each propagation can result, on the average, in approximately nine pin errors. In comparison, there are approximately four pin errors resulting from the first-order propagation.

3.1.5(b). External pin error distribution

This subsection quantifies the impact of within-chip transients on the external environment through I/O pins. Three examples of the error-probability pattern at the external I/O pins are shown in Figure 7. Note that transients at each

The error signatures can be used to insert realistic error patterns to study the impact of different with-in chip failures at the board and higher levels.

3.1.5(c). Charge level analysis

Statistical analysis of the the error data was performed to determine the effect of different charge levels in the injected transients on the severity of latch, pin and functional errors. Figure 8 shows the frequencies of latch, pin and functional errors as functions of the charge level in the injected transients. Note that beyond 7 pC, the number of error occurrences remains relatively constant; that is, an additional charge does not result in an increase in the error probability. At this charge level essentially all the latches in the propagation path have been affected.

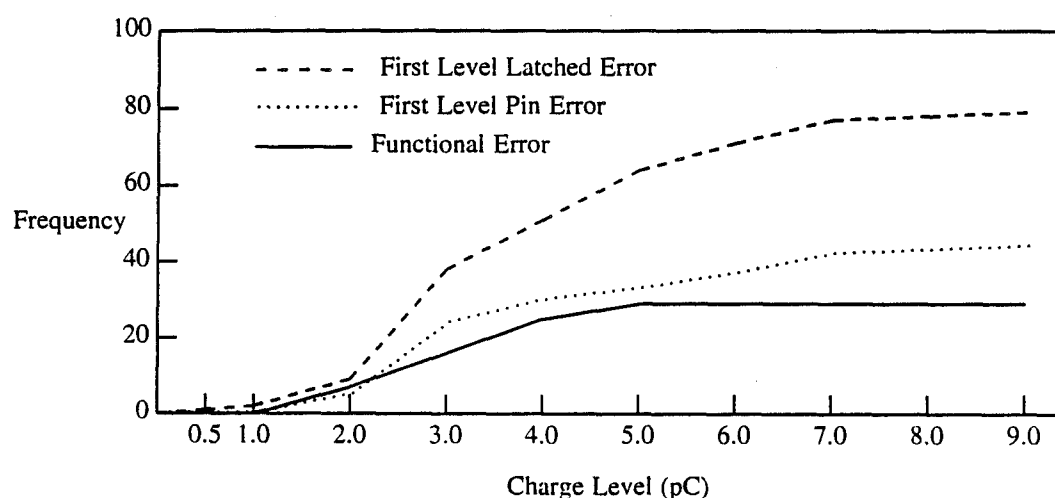


Figure 8: Error frequency by charge level.

For latch and pin errors, there is a charge threshold of 2 pC at which a sharp increase in error activity occurs. Over 95 percent of the latch errors occurred at charge levels greater than 2 pC, and 100 percent of the pin errors were observed for charges at or above 2 pC. For functional errors, however, the threshold is not so well defined, which is probably due to the fact that functional errors can also result from second- and higher-order latch errors (in addition to being caused by the first-order effect of a transient). The higher-order effects, of course, are not charge dependent, hence a charge threshold does not occur. Figure 9 shows the frequency of second- and higher-order latch errors and the functional upsets. Note that the frequency of the second- and higher-order latch errors also lacks the distinctive charge threshold.

Figure 10 shows, for each functional unit, the first-order latch and pin error distributions by the charge level in injected transients. For charge levels above

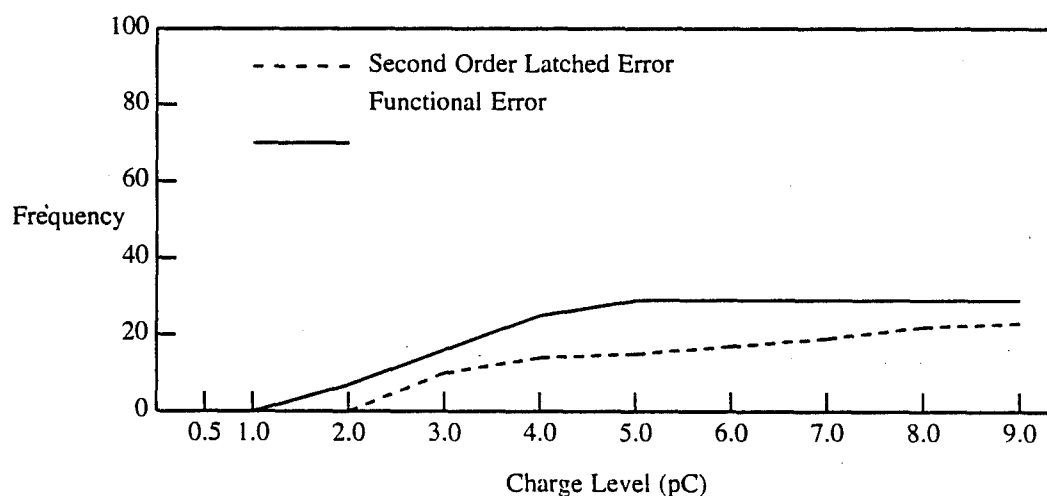


Figure 9: Second-order and functional errors.

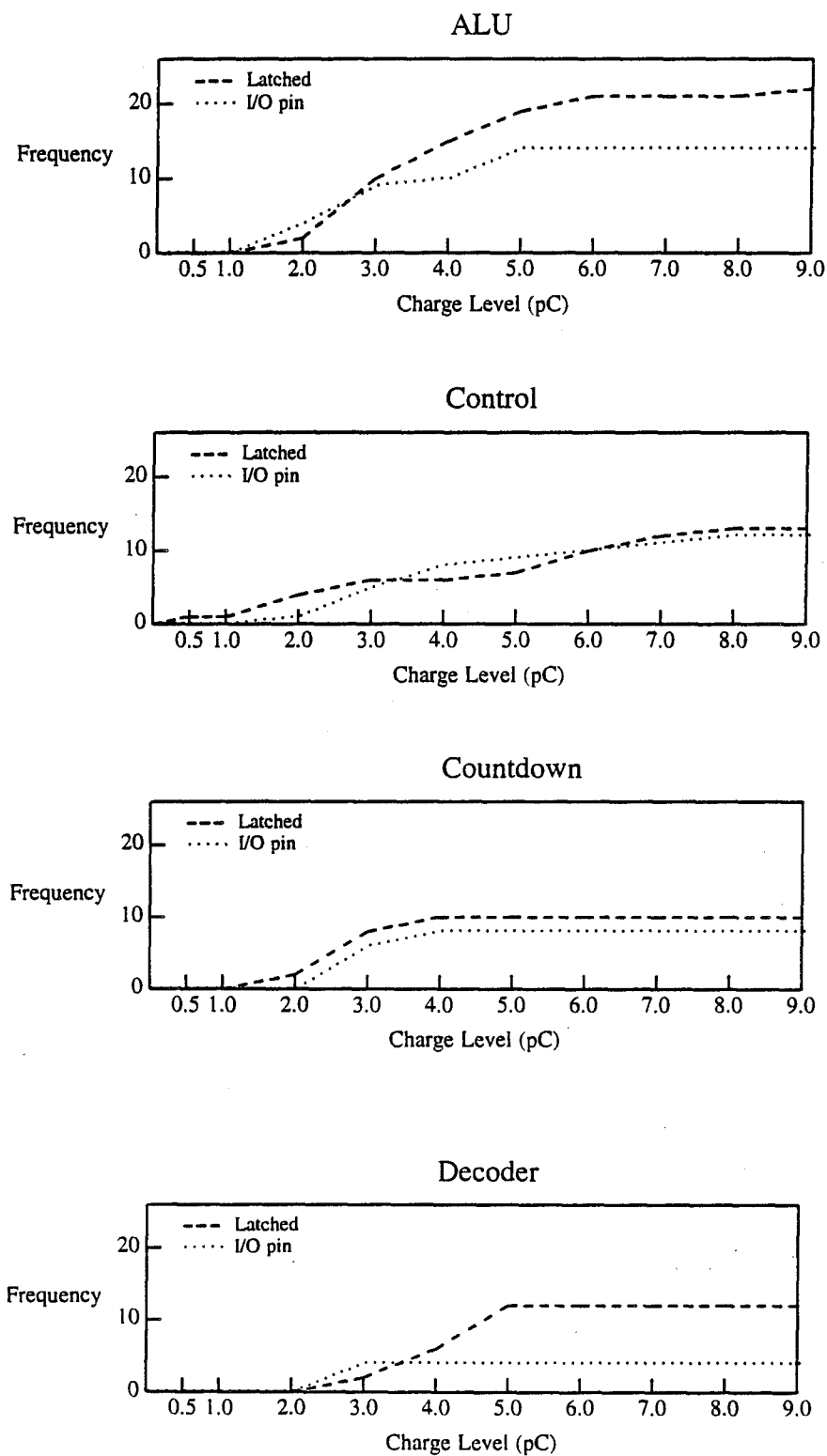


Figure 10: Error distribution by charge level.

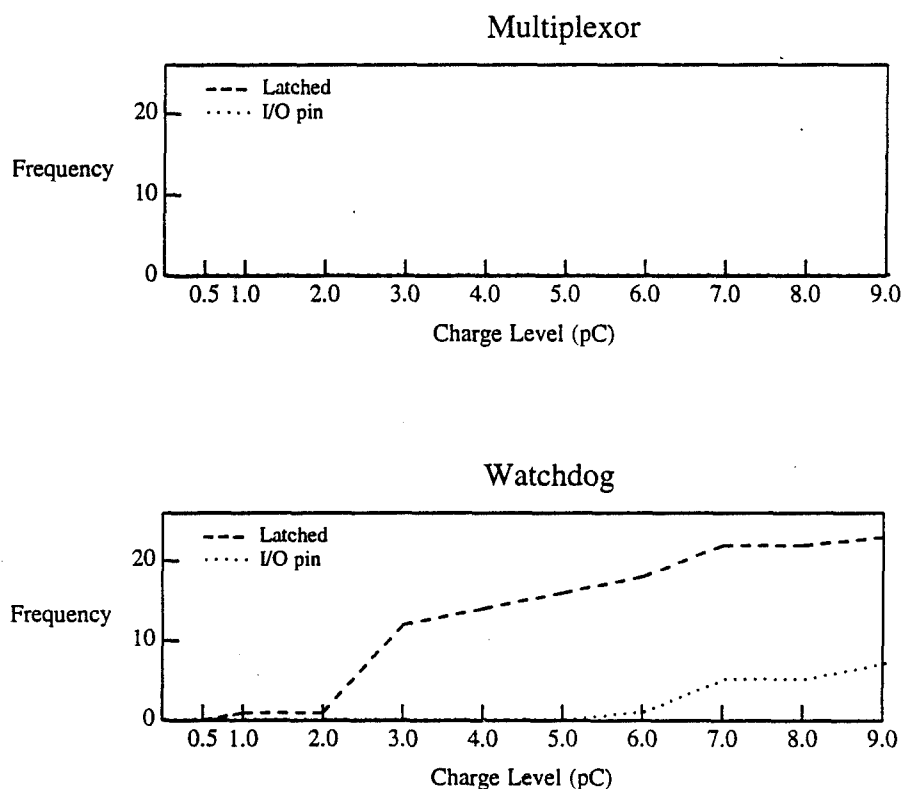


Figure 10: Error distribution by charge level (continued).

the threshold, the ALU and the watchdog units have the highest probability.

The watchdog unit had high latch error occurrences, but pin errors occurred only for charges above 6 pC. The reason is that, although an error can quite easily be latched in the numerous feedback paths in the watchdog, it does not propagate to the external pins. The decoder unit showed a relatively low pin error propagation probability.

The chance of transients below the threshold being latched is generally small, except for those in the control unit. In the control unit, the possibility of having latch errors is high, even at 2 pC and 3 pC. The relatively small capacitive loading of the feedback paths to the latches in the control circuit explains

this low charge. The relatively small capacitive loading of the feedback paths to the latches in the control circuit explains this low charge sensitivity.

As shown in Figure 10, the multiplexor does not have any latch or pin errors, because the electrical nodes in the multiplexor unit have high capacitances due to their large number of fanouts.

3.1.6. Empirical models

Fault propagation usually occurs because errors can be latched and then can migrate to different sections of the chip. A latch error can stay latent and undetected until it migrates to the pins at a later time. The additional internal propagation between latches can increase the probability of generating functional upsets. Thus, a characterization of the latch-to-latch fault propagation patterns is important. A latch error can either relatch, propagate out to the I/O pins and/or disappear in each clock cycle.

3.1.6(a). Error propagation model

The propagation of the latch errors in time (in clock cycles) for the control unit is illustrated in Figure 11. In this figure, the x-axis represents the clock cycles from the fault injection time, and the y-axis represents the total latch error count for each clock cycle. It can be seen that, given a certain number of latch errors in the first clock cycle, the number of latch errors degenerates significantly until the fourth clock cycle. At approximately the fifth clock cycle, the number

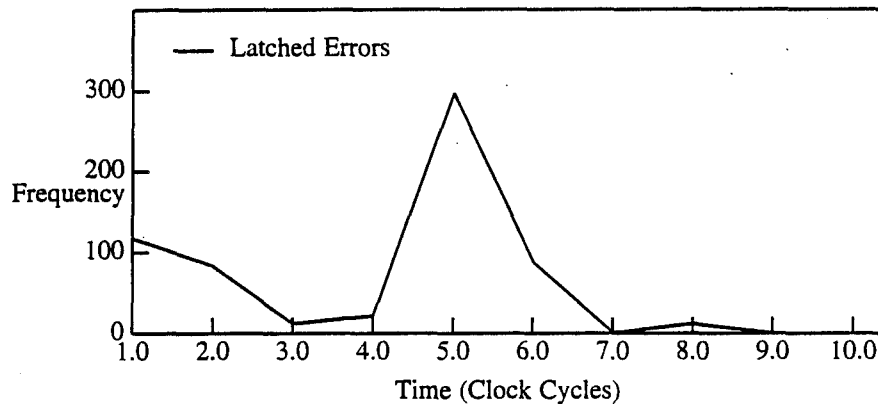


Figure 11: Latch errors in time.

of errors rapidly multiplies. Thus, despite the fact that the latch errors last until the fifth clock cycle on only a few occasions, when they do last, the number of errors is large, which occurs because at this time period, the error signal enters a unit with a large number of latches and high fanout, e.g., the ALU registers. After the sixth cycle, the number of errors degenerates significantly until it finally disappears after the eighth cycle. Thus, the impact of latch errors lasts at the most up to eight clock cycles from the time of fault injection.

For analysis purposes, the clock cycles in which the number of latch errors increases in comparison with the previous cycle are defined as *error explosion* cycles. Clock cycles in which the number of errors decreases in comparison with the previous cycle are defined as *error degeneration* cycles. In Figure 11, an error explosion occurs in the fifth clock cycle. The chance of functional errors and pin errors occurring may be maximal during this period of error explosion. In the sixth clock cycle (a degeneration cycle), the number of latch errors decreases to about one-third of the number in the fifth clock cycle.

A model depicting the process of error explosion and degeneration that result from a transient fault for the overall system is shown in Figure 12. The model is derived by averaging the results obtained for all the injected faults. As illustrated in the model, an injected fault either becomes latched (represented by *latch-error* state) or has no impact on the circuit (represented by *fault-free* state). The *explosion* state represents the situation where the number of latch errors in the current clock cycle is greater than that in the previous cycle. The *degeneration* state represents the opposite scenario, i.e., the number of latch errors decreases. In each clock cycle (either the degeneration or the explosion state), the number of latch errors are monitored and averaged for each state. The value assigned to a state is the average number of latch errors in that state. State

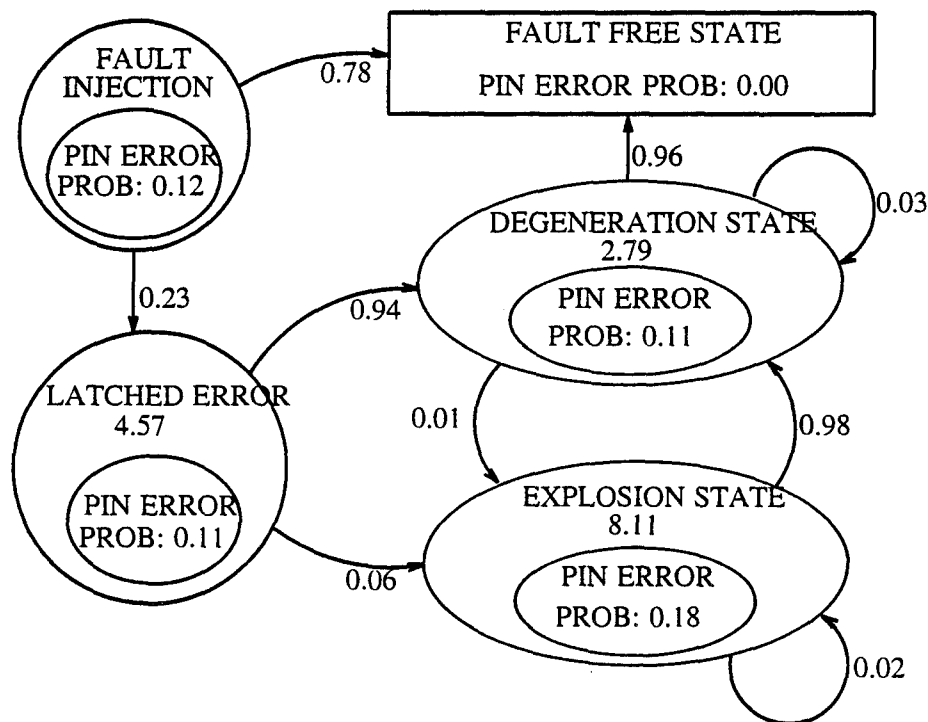


Figure 12: Error explosion/degeneration model.

transitions occur at each clock cycle. The transition probability from state i to state j is calculated by the ratio of the number of transitions from state i to j , to the total number of transition out of state i .

From Figure 12, given a transient fault, there is approximately an 80 percent chance of there being no impact on the chip. Although the chance of a latch error resulting in an error explosion is small (six percent), when it does occur the average number of latches holding an erroneous value is large (8.11 latches), i.e., although the explosion event rarely occurs, is potentially disastrous. The chance of latch errors, in the explosion state, causing pin errors is higher than that for the degeneration state (18 percent compared to 11 percent). This is clearly true because, with the larger number of latch errors, the chance of error propagation to the pins is increased. After an explosion, there is a 98 percent chance of latch errors degenerating and then becoming fault free.

In summary, the probability of a sustained explosion is very low (2 percent). The chance of uncontrolled propagation is small, thus the overall impact of a transient is well contained. Further, if no latch error occurs within eight clock cycles, no significant damage is likely to happen.

3.1.6(b). State transition model

The foregoing section presented an analysis of the forward propagation in time of an injected transient. This section examines the question: given a latch error in a unit, where did it come from? The question of the internal module-to-module latch error propagation is addressed. It will be shown that the results of

this analysis are useful in identifying several critical aspects of the system. Some examples include the identification of the critical error-propagation paths, the determination of the module most sensitive to fault propagation, and the module with the greatest potential for causing external pin errors.

Figure 13 shows a state-transition diagram, based on the measured data, to quantify the inter-module latch-error propagations. In the figure, the states represent error conditions in the specified functional units. Note that the model is

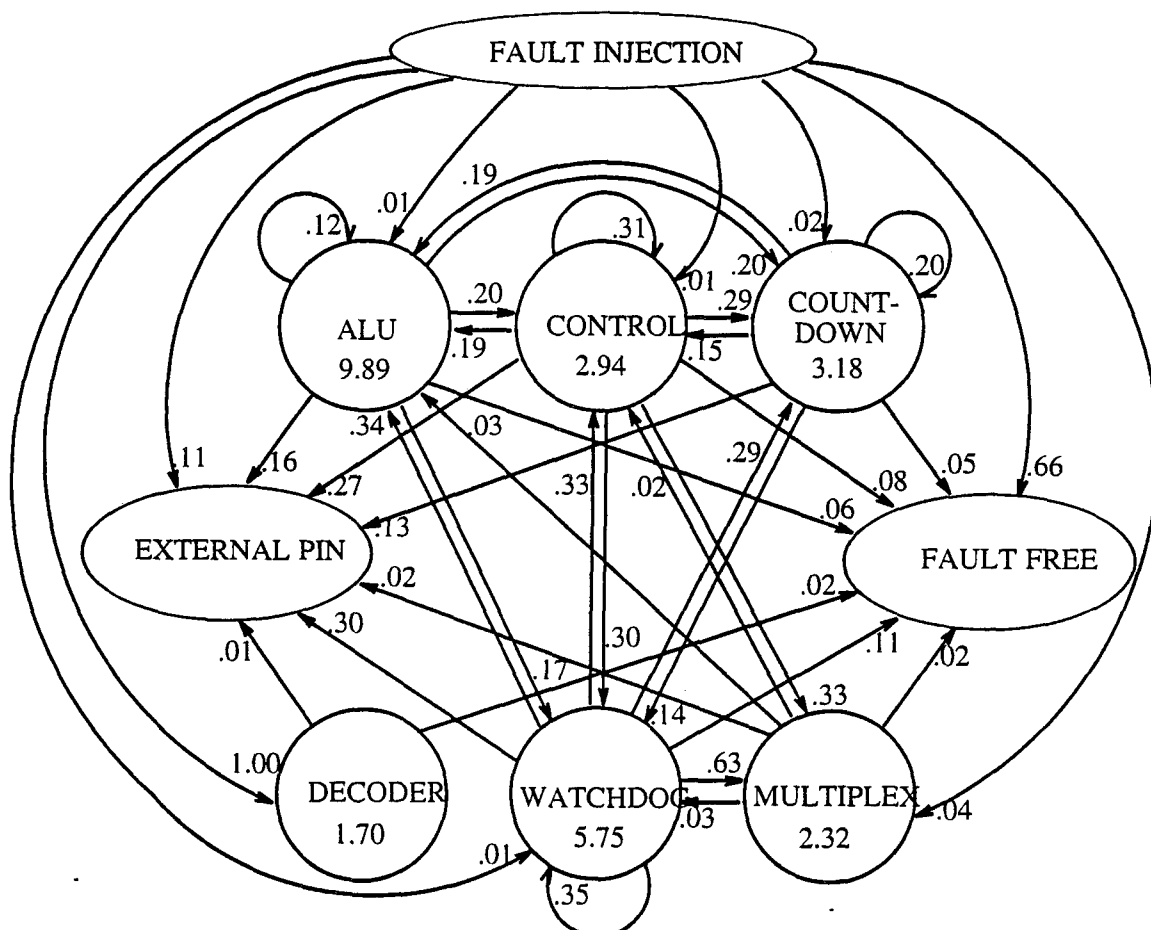


Figure 13: State transition model.

of the inverse Markov⁷ type. Thus, given a latch error in a specified unit, the model shows the probabilities that each of the other functional units are the error sources. For example, in the figure, given an external pin error, the probability of the ALU being the error source during the previous cycle is 0.16; the probability of the control unit being the error source is 0.27. The transition probabilities were calculated by monitoring latch errors in each functional unit for each clock cycles. In order to designate source of error, the path(or interconnects) of error propagation and the faulty state of the system in the previous the clock cycle are identified. The value assigned to a state is the average number of latches affected by transition into that state.

The model shows that latch errors in the decoder unit do not propagate to other functional unit. All latch errors that occurred in this unit were due to the direct effect of the injected transient, because the latches in the decoder unit are well isolated from the inputs of the other units. The probability of a latch error in the decoder unit propagating to the pins is small (0.01). Thus, the decoder is not a critical unit from a fault-propagation point of view.

The model addresses several issues raised at the beginning of this section. For example, the model shows that the critical fault-propagation path in the system is between the control and the watchdog units. Given a latch error in the control unit, the probability that it propagated via the watchdog unit is 0.33. Conversely, the probability of the control unit being the source for a latch error

⁷The usual *forward transition* Markov model cannot be used in a simple way to describe latch-error propagation since a latch error can propagate out to multiple locations at once, i.e., a latch error can propagate to both the external pins and other latches in

in the watchdog unit is also high (0.30). When the other units are examined, it is seen that, although the one-way propagation probability is high in some cases (e.g., 0.63 from the watchdog unit to the multiplexor), none has a higher two-way propagation probability. The critical path resulting in such a *loop* has a greater chance of resulting in an uncontrolled error explosion. Therefore, all other factors being equal, the best way to reduce intermodule error propagations is to protect the interconnections between the watchdog and the control units. Since a significant number of functional errors result from the second- and higher-order latch errors, the system-level impact of providing this protection is expected to be a decrease in the probability of functional errors.

The model also shows that the module with the highest potential to cause external pin errors is the watchdog unit. Thirty percent of all pin errors were due to the latch errors in the watchdog unit. Hence, to reduce the number of pin error occurrences, the outputs of the watchdog unit should be protected. The module most sensitive to fault propagation is the ALU unit. Of all the functional units, an error occurrence in the ALU will most likely lead to the largest number of latch errors (9.89). Applying internal retry to ALU operations may be a successful way of reducing the number of latch errors.

Finally, it is seen that the probability of an injected transient directly causing pin errors is low. More than 90 percent of the pin errors are due to second- and higher-order propagation from latch errors. Similarly, the probability that an injected transient will directly cause a latch error is also low. Notice that less

the circuit. Thus, an inverse Markov model is used to describe the transition from the consequent error source in each state.

than five percent of the latch errors for each state are due to the direct propagation from the injected transients. The fact that over 95 percent of latch error occurrences are due to propagations from other latch errors makes fault propagation a critical issue from a reliability perspective.

3.1.7. Discussion

This section described a mixed-mode simulation approach, which simulates the fault sensitivity of a chip-level design. The approach can effectively evaluate alternative tactics at the design stage where changes can be made at low cost. Faults are automatically injected in runtime at the device level, and their propagation and impact are monitored at the gate and function levels. A number of techniques for fault-sensitivity analysis have been proposed and implemented in the mixed-mode simulation approach. These include: transient impact assessment on latch, pin and functional errors, external pin error distributions due to within-chip transients, and error propagation models to depict the dynamic behavior of latch errors.

A design analysis was illustrated through a case study of the impact of transient faults on a microprocessor used in jet-engine control. The study was used to identify and isolate the critical fault-propagation paths, the module most sensitive to fault propagation, and the module with the highest potential for causing external pin errors. The fault-propagation path between the control unit and the watchdog unit was seen to be the most critical, which indicated that an increase in the fault tolerance of this link may significantly improve the system

dependability. The watchdog unit was seen to have the highest potential for causing external pin errors. An error occurrence in the ALU is more likely to lead to the largest number of latch errors than one in any other functional unit.

We emphasize that the results of the case study are specific to the target system. The methodology however, is general and can be used to evaluate other fault-tolerant designs.

3.2. Fault-Dictionary Based Approach

This subsection presents a methodology for performing fast transient-fault simulation. The methodology captures the logical error pattern that results from a device-level transient and propagates the resulting error using concurrent simulation. Several levels of fault dictionaries are used to record the error patterns that result from device-level transient faults. Although fault dictionaries have been used in the past to test and diagnose permanent failures, e.g. stuck-at faults [48] and bridging faults [49], the application to transient faults creates additional significant problems. In our approach, gates around the fault-injection region are extracted, and a subcircuit consisting of these gates is formed. This subcircuit is exercised by exhaustively applying input combinations while performing fault injections. The resulting error patterns at the output of the subcircuit and the corresponding input vectors are recorded. The recorded error patterns are used to inject logic errors, at runtime, in the entire circuit. The approach makes use of *importance sampling* to reduce the number of fault injections necessary to obtain statistically significant results. The presented methodology is illustrated using the MC68000 microprocessor.

Figure 14 depicts the overall experimental approach. The approach takes as input a net list of the hardware description, timing information and test software. A device-level fault behavior dictionary is generated by extensive circuit-level fault simulations. The electrical-level simulation engine used is SPICE [50]. The recorded error patterns are used to inject logic errors concurrently at runtime. The effects of injected logical errors are propagated throughout the design using a

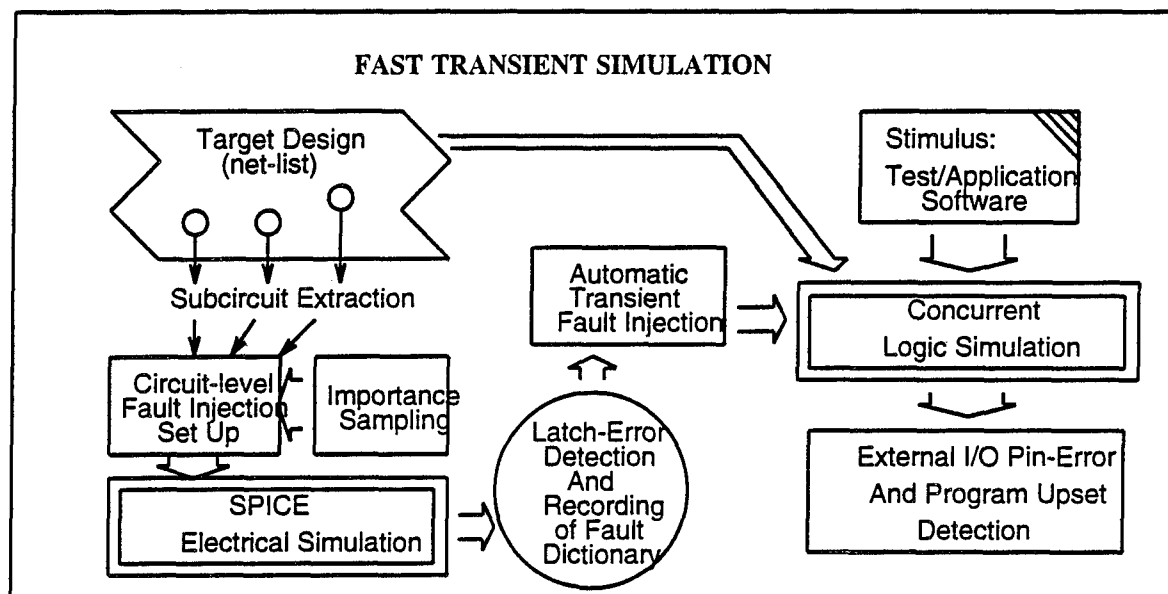


Figure 14: Experimental approach.

concurrent simulator. Errors affecting the external environment are monitored at the I/O pins. The next subsection describes the details of the fault-dictionary approach.

3.2.1 Circuit-level fault injection

The aim in this subsection is to generate *logical* error-behavior patterns of smaller subcircuits and store them on a fault dictionary in a form that is as compact as possible. Later we can inject, concurrently in runtime, the generated logical error pattern/behavior on the entire target design.

Combinational subcircuits, in acceptable sizes for a large number of repeated SPICE simulations,⁸ are chosen for the fault injection for the device-level fault

⁸Typically, thousands of repeated simulations of the subcircuit are necessary to try out all the input combinations and to inject faults on every node for each input vector. Experimental results indicate the size of the subcircuit should be limited to about 50 transistors.

dictionary generation process. The target subcircuit and latches driven by the outputs of those locations are extracted and evaluated. The extracted subcircuits are exercised with exhaustive combinations of inputs while fault injections on each node are performed. Faulty behavior at the outputs for each subcircuits is analyzed and recorded in the dictionary according to the subcircuit's input vector, fault-injection time, and location.

We also address the issues of timing, asynchronous logic behavior and interfacing the logic-to-electrical and back to logic-level at the analysis-mode boundaries following the techniques proven in [22].⁹ The steps involved in the fault-dictionary generation process include: subcircuit extraction, transient-fault injection, fault-propagation and latching, and recording of fault dictionaries.

Subcircuit extraction: Randomly chosen subcircuits are extracted from the VLSI design. An extracted subcircuit contains gates in all the fanout cones (paths) from a combinational region and their respective terminal latches. Figure 15 shows an example subcircuit extracted from the ALU of the MC68000. The inputs i_1, i_2, \dots, i_8 are the fanin inputs of the subcircuit (Subckt-C), and all the fanouts of this circuit are terminated by latches N17, N18, and N19. Each input is synchronized to one of the system clocks (C1-C4) as shown in the timing diagram. The reasons for setting up Subckt-C with all of its I/O lines

⁹For example, the issues in the accuracy of the signal transfer, between the electrical-level and the logic-level analysis, are resolved. Experiments were conducted in [22] to determine the size of circuit to be simulated at the lower level, for fault injection, in mixed-mode simulation. For a transient fault-induced voltage in a digital circuit to stabilize, a signal must travel through a sufficient distance in the circuit. To ensure that this requirement is met, we simulate in electrical level the entire fanout path, until it is terminated by a latch driven by system clock or other sequential signal. Once a fault gets latched, we can perform the rest of the simulation at the logic-level analysis, since only the 'logic state' of the machine has to be considered.

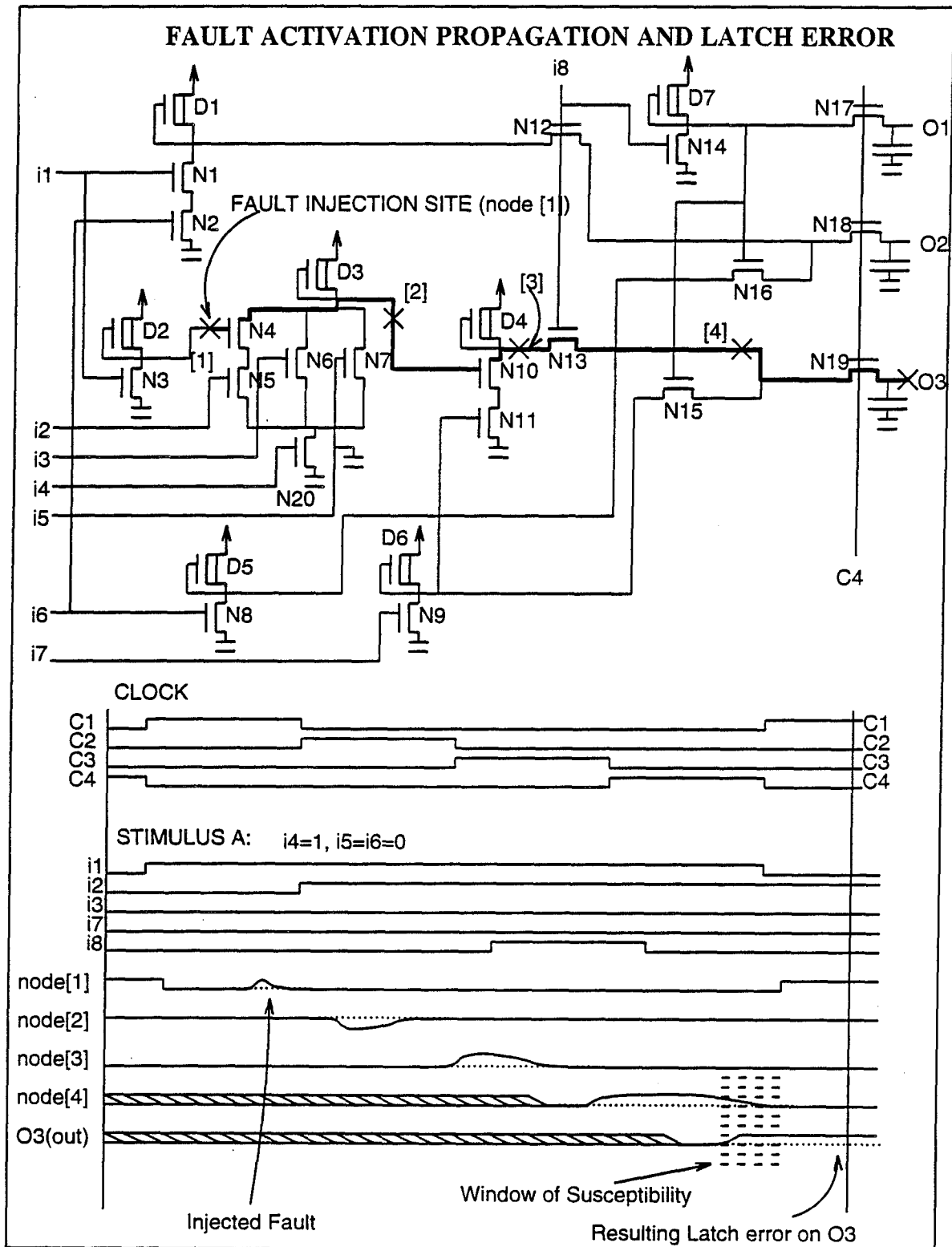


Figure 15: An example of subcircuit (Subckt-C).

synchronized to the system clocks are to ensure controllability of the input timing and to determine when to observe the output during the circuit/timing simulation.

Transient fault injection: For each node in Subckt-C, a number of current-transient injection/simulation runs are performed. In our experiment a current-transient with the charge level of 8 pC was chosen for fault injection. This charge level was chosen because smaller charge transients are unlikely to cause any logic changes, and higher charge transients have no significant additional effect [22]. In each run, a fault is injected at a specific time during the simulation period. The resolution of the fault injections in time is chosen experimentally: if the fault injection rate is too small, no latch-error is likely to occur; if the rate is too high, an unacceptable number of simulations is necessary. We use an importance-sampling approach [51] to increase the chance of a latch error through biased sampling of fault-injection time-points. The objective of importance sampling is to concentrate the distribution of the sample points on the parts of the interval that are of the greatest interest, instead of spreading them uniformly. First, using gate delays, we estimate the time point at which a signal propagation ends up at the latch. Then we perform fault injection with a higher distribution of samples at or near this time point. For each fault-injection time-point, for a given node, an exhaustive set of input stimulus patterns is applied.

Fault propagation and latching: An injected fault can either propagate or be masked. A case wherein a fault propagates and becomes a latched error is illustrated in Figure 15. When a fault occurs (node N1), for a given input vector

(stimulus A), it propagates to the latch(es) with a certain probability. In the example, the faulty signal propagates from node N1 through nodes N2, N3 and N4 and gets latched at the latch N19. The fault propagation path is shown by the bold line in the circuit diagram. Note that propagation does not occur if the circuit is not in a favorable state, i.e., if input (i8) is zero, then the faulty signal does not propagate from node N3 to node N4. The transient propagates only when masking does not occur. For example, the simulation data from Subckt-C indicates that the probability of fault propagation is 0.213. Once a fault propagates to a latch, the faulty signal can become latched if it is clocked. For Subckt-C, the probability that a propagating transient is latched is 0.0869. The overall probability of a fault becoming a latched error is given by:

$$p(latch_error) \equiv p(fault_propagation) \times p(latching).$$

For Subckt-C the probability of having a latch error is 0.00245.

Fault dictionary: Fault behavior is recorded at several levels of detail ranging from the detected fault patterns to the probability values of possible fault patterns for a given fault. A fault dictionary is a recording of the faulty behavior at the output of a subcircuit. Three levels of fault dictionary are proposed. There is a trade-off between the accuracy of probabilities of error patterns generated by a device-level fault and the size of the dictionary.

Level-I: In a level-I dictionary, the faulty behavior at the outputs for each subcircuit is analyzed and recorded in the dictionary according to its input vector,

fault-injection time, and location. Figure 16 shows an example of a level-I dictionary. For example, for an input vector (0,0,0,0,0), fault-injection node N2, and fault-injection time (5 nsec), the dictionary shows which output is altered due to a transient. Following the bold line from the input condition to the error-pattern condition shows that the output O1 is altered by the fault. This dictionary is a direct recording of all device-level fault simulation performed and can require enormous amounts of a storage space. To cut down on the size, Level-II and Level-III dictionaries are proposed.

Level-II,III: In a Level-II dictionary, faulty behavior is recorded according to its input vector and fault-injection location. Each entry, corresponding to a faulty

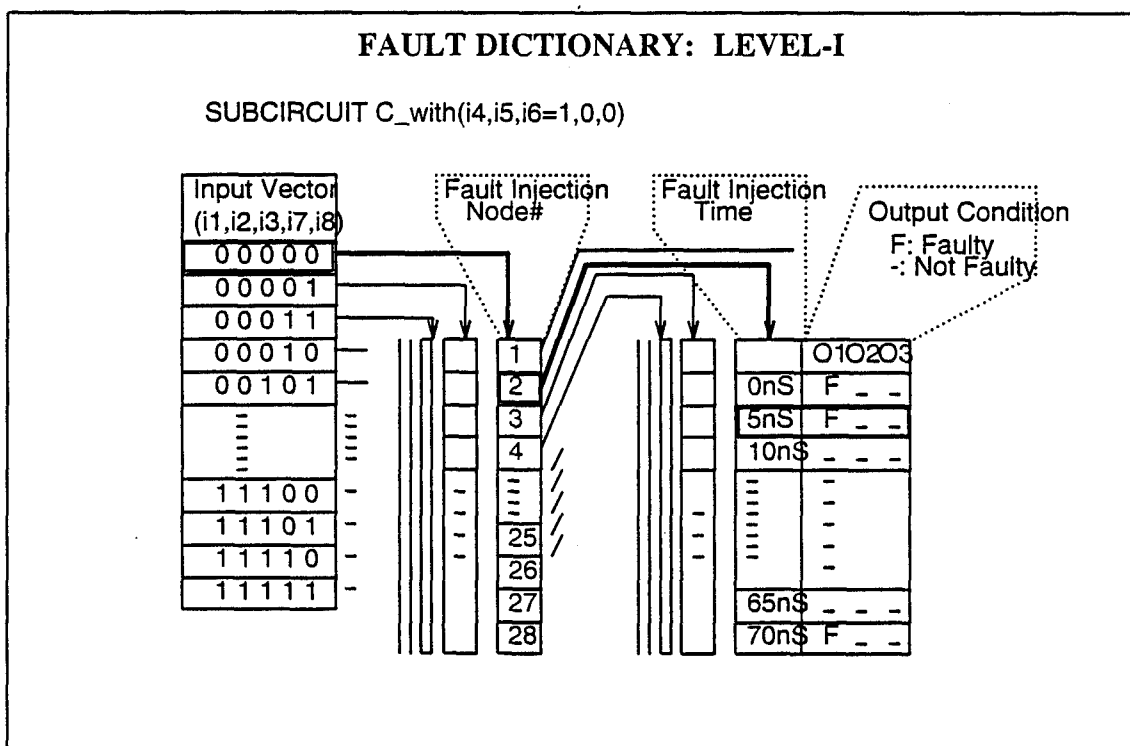


Figure 16: Level-I fault dictionary.

output pattern, contains the probability that the given pattern occurs from an injected transient. Figure 17 shows the level-II fault dictionary generated for Subckt-C. For example, for an input vector (0,0,0,0,0) and fault-injection node N3, the dictionary shows that there is a 97.7 percent chance of having no error, a 1.1 percent chance of having a single-bit error (output O3), and a 0.8 percent chance of having a multiple-bit error (outputs O1 and O3). The Level-III dictionary is a reduced version of the Level-II, further reducing the required storage space. In a Level-III dictionary, a faulty behavior is recorded according to its input vector only. A subcircuit is treated as a black box at this level; the dictionary contains, for a given input vector, the probability of each possible error pattern.

Figure 17 shows that the probability of having latch error due to transients is very small. Over 99 percent of faults injected in this experiment do not get latched, and hence do not cause any damage. Less than one percent of the injected faults get latched. Although the probability that a transient will get latched is small, recent system-level measurements show that the actual number of transients that occur in a system can be quite large [24]. Hence, the absolute number of latched errors can be significant. Further, for mass-produced chips, the absolute number of errors per product can also be large. What we are concerned with is the effect of those faults that get latched. We calculate the conditional probability of a specific error pattern given that injected transients get latched. The last column in Figure 17 shows the conditional probability for each error pattern.

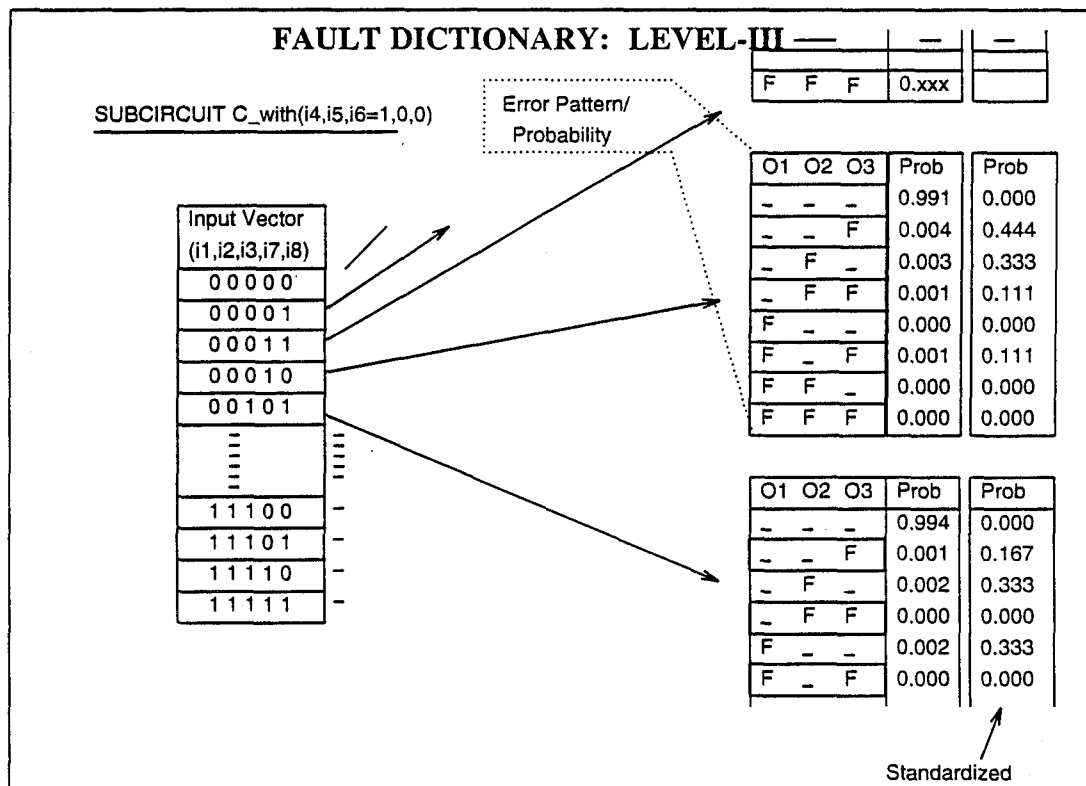
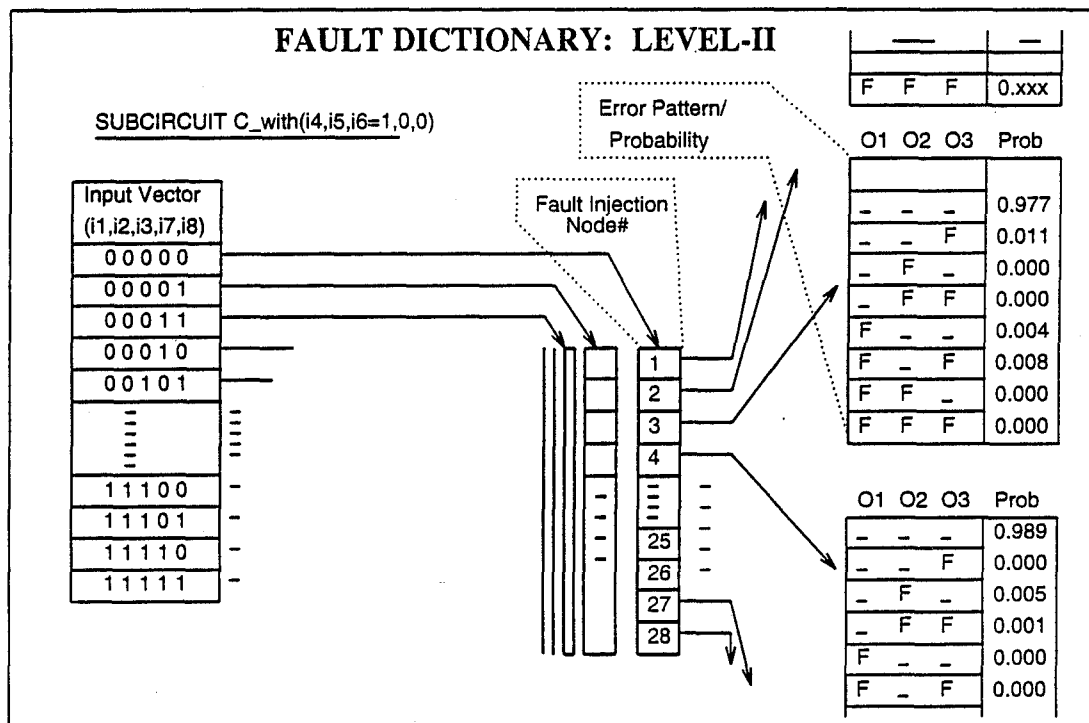


Figure 17: The Level-II and Level-III fault dictionaries.

3.2.2. Concurrent transient simulation

To achieve acceptable simulation speed for analyzing such large systems, we employ a concurrent fault-injection approach based on CHAMP [52]. Concurrent simulation allows simultaneous evaluation of a great number of faults in single simulation pass. CHAMP, a concurrent logic simulator developed at the University of Illinois, has been shown to perform ultrafast fault simulation of very large chips.

In order to accurately map the device-level fault behavior at the logic level, we tied the fault dictionary generated with SPICE simulation to CHAMP. Thus, for a single fault, the concurrent simulator simulates the circuit until the fault injection occurs. The fault-injection process will be implemented as a runtime modification of the logic state in the circuit whereby an injected-error pattern is supplied by the device-level fault dictionary. The resulting logic-signal changes in the subcircuit simulated with SPICE are due to the device-level transient faults and are fed into CHAMP as logic-level errors. The concurrent simulation of the entire circuit continues.

In concurrent fault simulation [53] all the faulty machines are simulated in a single pass together with the good machine. To avoid duplicating the circuit description, only the differences between a particular faulty machine and the clean one are recorded at any particular time. This is achieved by associating with each node in the circuit the state of the good machine at that node and a list of fault effects (states of faulty machines) for those faulty machines in which the

state of the node differs from its state in the good machine. This list is called the fault effects list.

Concurrent fault simulation is based on the event-driven simulation paradigm [48] where a change in the logic value of a node (in the good or the faulty machine) constitutes an *event* and causes that node to be placed on an *event queue*. The simulation progresses through discrete time steps by handling all the events at the *current time* and then advancing the simulation clock. Simulation starts by applying a vector to the primary input nodes of the circuit, which causes a subset of these nodes to be placed on the event queue. When an event is removed from the event queue, it is processed as follows:

- (1) If the event results from a change in the state of a node in the good machine (good event), then all the elements (gates or subcircuits) having that node as input are evaluated. A change in an output node of any such element causes that node to be scheduled at the appropriate time (the current time plus the delay of the element).
- (2) An event from a faulty machine (faulty event) is handled similarly with the state of that node taken from the fault effect list.
- (3) When evaluating an element activated by a good event, any fault effect on the input nodes of the element is propagated to the output, if the fault causes the state of the output to differ from its fault-free value.

- (4) If the state of a node in the good machine becomes identical to that in a faulty machine, then the corresponding fault effect is dropped from the fault effect list on that node.

The advantage of concurrent fault simulation is its speed, which results from only the active faults in the circuit being considered. However, if the number of active faults is relatively large, then the speed degrades due to the overhead incurred from the maintenance of the fault effect lists [52]. Another drawback of concurrent fault simulation is its unpredictable memory requirements.

3.2.3. Illustration of the fault dictionary approach

The fault-dictionary approach is illustrated with the MC68000 microprocessor. Six subcircuits are extracted from different parts of MC68000 and are analyzed. The extracted subcircuits are exercised with exhaustive combinations of inputs while fault injections were performed. Fault-injection time resolution was 1 nsec or less. Generation of a fault Level-I dictionary for each subcircuit took on the average about 27 hours on a SUN SPARC ELC workstation. Generation of Level-II and Level-III dictionary from the Level-I dictionary took only several minutes, since it only involves compressing probabilities in each entry. A total of 223,360 SPICE simulations was performed. Faulty behavior at the outputs for each subcircuits were recorded in the dictionary according to its input vector, fault-injection time, and location.

The generated logical error pattern/behaviors from the Level-III fault dictionary were injected concurrently at run-time on the target design. Over 30,000

error patterns were injected into the microprocessor and 48 (0.1548%) of those injected errors propagated to the external I/O pins and were detected. Average error detection latency was 4.75 clock cycles. Test vectors (stimulus) executed on the microprocessor during the simulation period included 75,000 instruction cycles.

Device-Level Results: The device-level fault injection results are shown in Figure 18. Out of 223,360 faults injected on six subcircuits, 427 (0.191 percent) were latched. The probability of fault latching in each subcircuit was less than 0.0025.

Figure 19 shows the latency distribution of latch errors. The X-axis shows the number of faults that were latched with various Y-axis latencies. Faults that have shorter latencies are closer to the latches. My results show that transients that occur near the latches have a greater chance of being activated. This result

Device-Level Fault Injection Result							
	#Inputs	#Latches/ Outputs	#Transistors	#Internal Nodes	#Fault Injection	#Latched	%Latched
Subcircuit A ALU LOGIC	5	3	36	31	39,680	83	0.209
Subcircuit B ALU LOGIC	6	4	31	27	69,120	101	0.14.6
Subcircuit C Byte Correction	5	3	27	22	28,160	69	0.245
Subcircuit D Data Bus Mux	4	2	22	21	13,440	29	0.216
Subcircuit E AOBL Latch	5	2	34	28	35,840	67	0.190
Subcircuit F Prog Counter Logic	5	3	31	29	37,120	78	0.210
Total	-	-	-	-	223,360	427	0.191

Figure 18: Device-level fault injection result.

was somewhat predictable, since the farther the node is away from the latch, the greater the chance it will be masked out during the propagation. These results indicate that any circuit-level fault-protection schemes should be focused on those gates near the latches.

Figure 20 shows the number of *bit-flips* (latches affected) in the event of fault propagation. Given that a transient results in a latch error, the chance of multiple errors is low (92 percent of the latch-error incidents were just single bit errors). The existence of multiple latch errors potentially can be a serious problem, since these errors can subsequently propagate and lead to multiple failures.

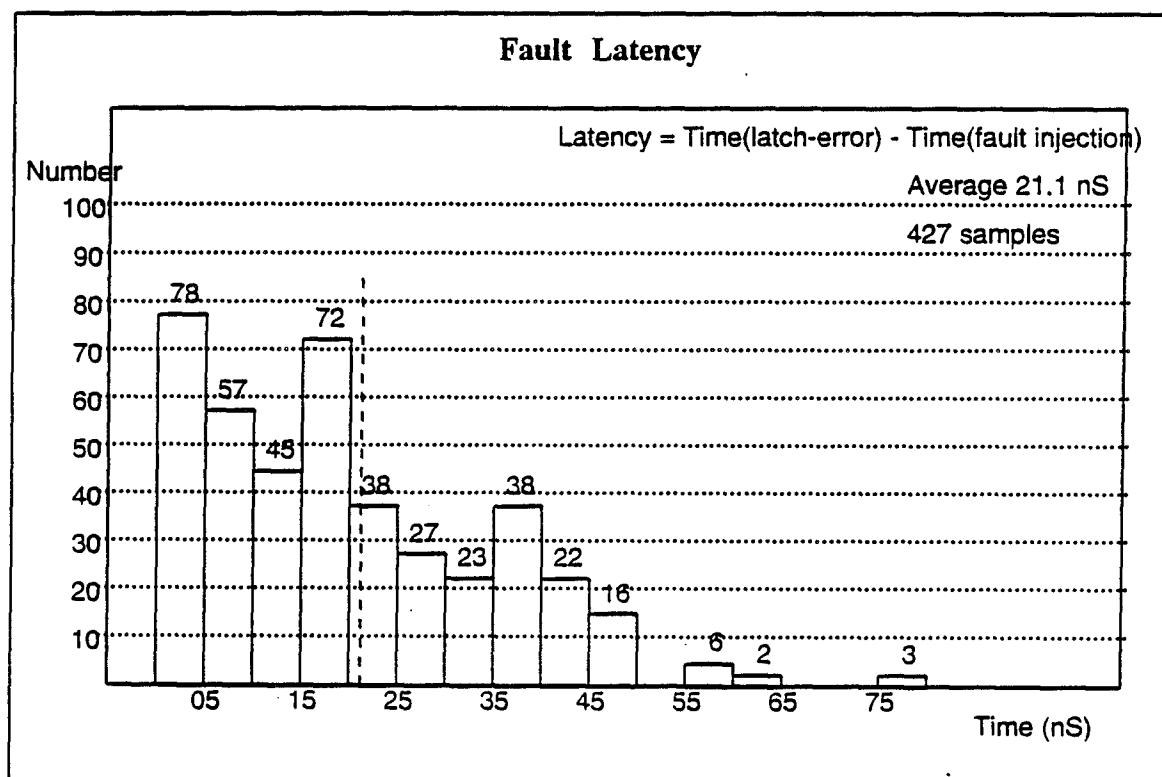


Figure 19: Fault latency.

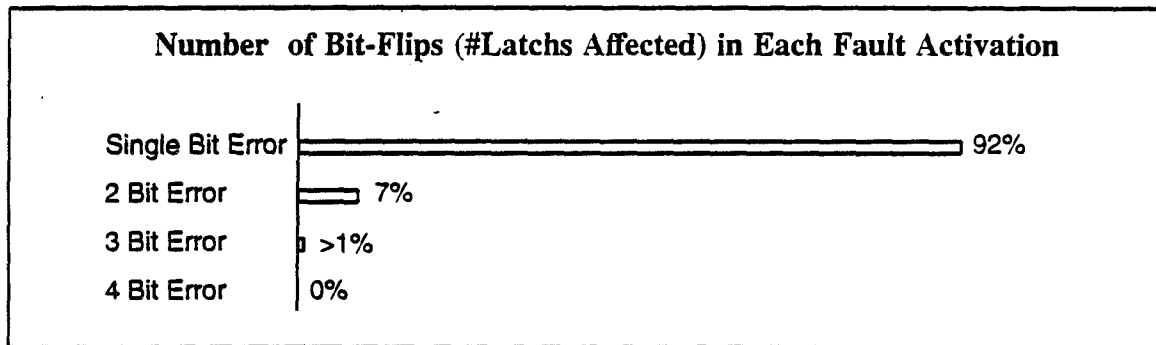


Figure 20: Number of latches affected.

3.2.4. Discussion

This subsection has presented an ability to generate fault behavior dictionaries for device-level transient faults. The gates around the fault-injection location are extracted, and a subcircuit consisting of these gates is formed. This subcircuit is exercised by exhaustively applying all input combinations while fault injection is performed. The latch-error pattern at the subcircuit output is analyzed and recorded in a dictionary. Entries in the dictionary consist of input vectors, fault-injection times and fault locations. In order to obtain a fault dictionary with statically significant values, we also develop a technique based on importance sampling to choose the fault-injection time point. The generated fault dictionaries are used to inject, in runtime, the logical pattern/behavior of device-level faults using the fast logic simulator. We find that 99.8 percent of the injected faults have no effect on the system, i.e., no latch error would result. Multiple errors are less likely to occur (8 percent), but they can become a serious problem, since they can cause multiple failures.

CHAPTER 4.

TRANSIENT FAULT MODEL

In simulating device-level transient faults, it is desirable to choose a fault model which is as accurate as possible. An important question to address is the level of detail with which fault models should be described in order to accurately evaluate the faulty behavior of the target system. Some simulation approaches choose higher-level fault models, such as the logic or system level, to gain an increase in simulation speed. However, while the simulation speedups of such models are often compared, rarely is the accuracy of the models ramified.

Several studies have investigated the impact of transient faults in computer systems through simulation. Transient faults are injected at several different levels of abstraction either by altering the logic values or by upsetting the current/voltage levels of the target nodes momentarily during the simulation.

The device-level studies reported in [4] and [5] accurately model transients using electrical-level simulation to capture the unpredictable circuit behavior under faulty current or voltage surges.

At the logic level, studies reported in [20], [11], and [23] implement transients as momentary *bit-flips* of the propagating signal. A register-transfer-level transient simulation, presented in [54], models transients as temporary changes in logic values in the memory elements of the simulated circuit. The corrupted values are either overwritten or propagated and cause errors in the other parts in

the system. In [55], a timing simulation technique that approximates the device-level fault waveform is proposed for a faster transient simulation. These approaches are significantly faster than device-level methods, since they do not rely on solving the circuit equations. However, there is no way of knowing how accurate their logic-level transient-fault models resemble the actual device-level faults. A transient can propagate along multiple paths and result in multiple latch errors. The chance of a faulty *pulse* propagating to a latch and becoming a latch error is a function of device-level parameters. Also, the shape of the *pulse* may be changed while it goes through different gates in the propagation path.

At the system level, transient faults can be emulated approximately by altering the logic state of the target system. This approach can be useful when a fast analysis speed is essential. A software testbed, REACT, that performs automated life testing of a variety of multiprocessor architectures through simulated-fault injections is reported in [29]. DEPEND, developed at the University of Illinois, exploits the properties of the object-oriented paradigm to provide a general-purpose, system-level dependability analysis tool that can evaluate various types of fault-tolerant architectures [28]. In [10], a testbed to perform fault injections and to monitor the impact on a target system has been developed. The testbed was used to validate a computerized interlocking system for the French railways.

Mixed-mode simulation approaches can be taken to combine the advantages of the accuracy of the device-level analysis and the speed of higher-level approaches. In the mixed-mode approach, device-level faults are injected in an electrical-level analysis tool like SPICE [50], and subsequent logic-level errors

are propagated at the gate and higher levels, which allows the target system to be simulated for a longer time period than does the device-level approaches described earlier. In [21] and [22], experiments to quantify the impact of transients from the device- to the pin-level were described. Transients with low charge levels (0.5 pC - 8.0 pC) were injected. The ensuing logic upsets and first-order latch and pin errors were analyzed through analysis of variance methods. Recently, a more efficient technique for performing transient-fault simulation has been developed [25]. The representation of a subcircuit which is subjected to transient-fault injections is dynamically switched among different analysis modes, i.e. electrical and logic levels, during the simulation.

4.1. Validation of the Mixed-Mode Transient Simulator

In using such mixed-mode simulations, the question of the accuracy of the signal transfer between the electrical-level and the logic-level analysis needs to be addressed. For a transient-induced voltage in the digital circuit to stabilize, a signal must travel through a sufficient distance in the circuit. In this context, we define the *gate distance* as the number of levels of gates between two nodes in the circuit. For example, Figure 21 shows 1,2,3 and 4 gate distances, from the fault injection location (marked X).

In order to determine the gate distance to be simulated at the electrical level, experiments were conducted wherein transients with different charge levels were injected into a randomly selected node. The HS1602 microprocessor was used for this experiment. Charge levels were chosen to span the entire range where

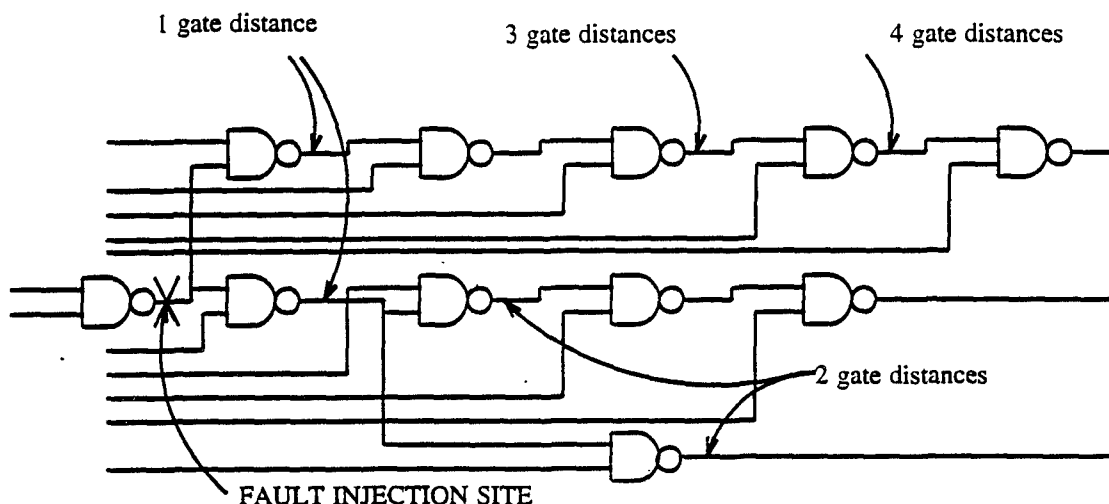


Figure 21: Gate distance definition.

logic upsets are likely to occur. This value was chosen experimentally to produce a worst-case deviation from the nonfaulted value without inducing permanent failures. Charge levels greater than about 10 pC can cause permanent latch ups (device failure) [47]. Initially, all gates within five gate distances from the target node were simulated at the electrical level. Next, the same injection was made with all gates within four gate distances from the target node, simulated at the electrical level. A logic comparison was made to verify the consistency between these two simulation results. Similar injections were performed with 3, 2 and 1 gate distances simulated at the electrical level, and again the logic comparisons were made. Experiments were carried out for several other randomly chosen nodes.

The above experiment was also repeated for transients occurring near latches. Transients near the latches have to be dealt with separately, because

latches (latches contain transmission gates) do not stabilize analog signals as well as other gates in CMOS technology. Five target latches in the circuit were randomly selected and the nodes up to five gate distances away from the selected latches were chosen as fault-injection locations. Analysis of the simulation results for different gate distances, simulated at the electrical level, was performed. Table 3 summarizes these results. For example, for combinational circuits, with a charge level of 7 pC, at least three gate distances from the point of injection have to be simulated at the electrical level. In general, for combinational circuits, up to three gate distances from the injection point have to be simulated at the electrical level. Transients occurring near latches require up to four gate distances be simulated at the electrical-level¹⁰.

Table 3: Minimum gate distances.

Charge Level	Combinational Circuit	Latch Distance (Gate Distance)				
		1	2	3	4	5
1 pC	1	2	1	1	1	1
3 pC	2	2	3	2	2	2
5 pC	2	3	3	2	2	2
7 pC	3	3	4	4	3	3
9 pC	3	3	4	4	3	3

4.2. Fault Model Comparison

For a large circuit, mixed-mode simulation still requires a large amount of both processing power and storage space, since each fault injection must be

¹⁰These numbers are representative of only the circuit under simulation. For other systems, our approach allows similar analysis to be performed to determine the appropriate gate distances that must be simulated at the electrical level.

treated separately. A method proposed to overcome the simulation resource exhaustion through a fault-dictionary-based approach is reported in Section 3.2. A fault-behavior dictionary generated from device-level fault analysis is used as a fast look-up table for a logic-level concurrent or parallel fault-injection simulation.

An important issue that has to be addressed is the level at which one inject faults to accurately depict the realistic fault behavior of a system, i.e., is it necessary to use device-level fault dictionary. Often, the analysis level is arbitrarily chosen. Rarely is there significant information on the relative validity of the models at different levels of abstraction. A measurement of the accuracy of a high-level fault model as compared to a low-level model is necessary to determine the accuracy of data resulting from the high-level model.

This section examines an accurate transient-fault model (analog current spike) and an approximate model (logic pulse) for studying transient faults in VLSI circuits. The examination is carried out by comparing the simulation results of the logic-level model to results obtained using a generally accepted circuit-level SPICE model. The logic-level model is represented as a discrete change in a signal level from logic zero to logic one, while the circuit-level model is represented as a double exponential current injection, shown to be an accurate representation of transient faults induced by alpha particles in [6]. The faulty responses predicted by the two models are compared at the system level; therefore, we may examine the difference in the predicted system behavior, and at the subcircuit level we may examine why these behavior differences occur.

The two models are simulated in SPICE, using a fault dictionary approach, so that the differences measured are due solely to the difference in the abstraction level of the two models and not to any differences in the surrounding simulation approach.

The comparison study was conducted in two phases. In the first phase, the faulty responses predicted by the two fault models were compared at the subcircuit level. To determine the subcircuit-level impact of the transient models, faults from both models were simulated at the device level using SPICE. The analog fault behavior was simulated as a double exponential current injection, the function proposed in [6]. The discrete model used to approximate the same behavior was a voltage pulse from zero volts to five volts with a chosen duration. The result of subcircuit-level impact of the transient models reported in [56] shows that, for some circuits, it is not possible to obtain same latch error with a discrete model as with analog fault model.

4.2.1. High-level impact of transient fault models

For the system-level comparison, we used a fault-behavior analysis approach based on a fault dictionary. The dictionary contains device-level fault-behavior patterns derived from SPICE-level injections.

Two types of dictionaries were generated: one using a realistic double exponential current injection to model the transients, the other using a discrete voltage pulse. In all other respects, the two types of simulation runs were the same. Thus, any differences in the results were due solely to the differences in

the fault models. Experiments were run with five different fault dictionaries. The first was generated using the analog fault model. The other four were generated using the discrete model with different transient durations (pulse widths). In most cases, discrete pulses with 2 nsec to 4 nsec widths closely matched the waveform generated by the analog model for our target system; therefore, fault dictionaries with pulse widths in this range were good candidates for analysis. The fault models used for generating the five fault dictionaries were:

A. Accurate transient fault model (device-level):

- (1) Double-exponential analog current pulse

B. Approximate models:

- (2) Discrete pulse with width = 2 nsec
- (3) Discrete pulse with width = 3 nsec
- (4) Discrete pulse with width = 4 nsec
- (5) Discrete pulse with width individually varied by injection location to fit the accurate model as closely as possible

A fault dictionary for the target subcircuit was generated using each of the above fault models. The generated fault dictionaries were used to inject logic errors at the subcircuit locations in the overall design. To map the device-level fault behavior/model to the logic-level analysis of the entire microprocessor, we tied the fault dictionaries generated using SPICE to a concurrent fault simulator. For a single fault, the concurrent simulator evaluates the circuit until a fault

injection occurs. To inject a fault, the logic errors corresponding to the error pattern supplied by the device-level fault dictionary are inserted into the circuit at runtime. Error patterns are automatically supplied by the fault injection facility for each clock cycle. The transient simulation uses a dynamic fault list where faults can be created or deleted any time during the simulation period unlike the fixed fault list used in stuck-at fault simulation. Dynamic management of the fault list is necessary since transients may occur any time during the runtime of the circuit and then may later disappear.

4.2.2. Results

System-level simulation was done by generating five fault dictionaries from the analog and pulse transient simulations described above: one for the analog model, and one each for the discrete model with 2 nsec, 3 nsec, 4 nsec, and variable duration (matched to the duration of the analog transient). A fault dictionary is a recording of all device-level simulation performed and can require an enormous amount of storage space. To reduce the amount of storage required for the dictionaries, Level-III dictionaries were used. In a Level-III dictionary, fault behavior is recorded according only to the subcircuit primary input patterns, assuming all injection times and locations are equally probable. A subcircuit is treated as a black box at this level. Thus, for a given input x , the fault dictionary will contain the probability of each error pattern that can occur under input x .

The generated fault dictionaries were used to perform concurrent fault simulation on the entire design. Over 5,000 error patterns were used to inject logic

errors for each experiment aimed at analyzing the validity of each fault model. The Level-III fault dictionaries were used to generate the error patterns used in the concurrent fault injection simulation. Test vectors (stimuli) executed on the microprocessor during each simulation period included approximately 130 instruction cycles. The external I/O pins and data registers were monitored, and detected errors were recorded. Table 4 shows, for each fault-model/dictionary: the number of injected error patterns/faults that resulted in latch errors (some injections resulted in pattern 000--no error); the number of detected external I/O pin errors; the average latency of detected pin errors; and the number of data error occurrences (an error in the value of any of the MC68000 data or address registers). The percent difference between each result and the accurate result of the analog fault dictionary is also given.

The resulting fault dictionaries are quite different, especially with regard to multibit errors. These differences are traced to differences in the slope and magnitude of the voltage of the transient error. Such discrepancies lead to a greater number of multibit errors in the analog simulation than in the discrete simulations. Furthermore, these multibit errors also lead to a greater number of data and external I/O errors. A significant difference (at least about 40 percent) in the number of detected I/O pin errors between the discrete model and the analog model was found. Both the error latency and the number of detected data errors were different by more than 50 percent between the best approximate model and the accurate model. Thus, discrete simulation gives a significantly lower number of detectable errors due solely to inaccuracies in its fault modeling.

Table 4: System-level impact.

CHIP-WIDE SCALE IMPACT OF TRANSIENTS						
pulse width	Discrete pulse waveform					Double-exponential Analog waveform
	2nS	3nS	4nS	Varied		
# Injected faults	70	221	615	796		1835
# I/O pin error	12	30	30	54		98
(%difference from analog transient result)	81.8%	69.4%	69.4%	44.9%		0.0%
Average latency of I/O pin errors	13.7 Clock Cycles	11.5 Clock Cycles	11.5 Clock Cycles	17.0 Clock Cycles		8.7 Clock Cycles
(%difference from analog transient result)	57.5%	32.2%	32.2%	95.4%		0.0%
# of data errors	11	41	41	65		147
(%difference from analog transient result)	92.5%	79.6%	72.1%	55.8%		0.0%

4.2.3. Discussion

Two transient fault models, an approximate logic-level model and an accurate device-level model, were compared to determine the inaccuracies inherent in the logic-level model due to its higher level of abstraction. The results of the comparison showed that the two models could be made to match by selecting an appropriate pulse width only if the injection site was a gate input or output with a single propagation path to the circuit outputs (or latches). However, if the injection site was a node internal to a logic gate, or if it had multiple propagation paths to the circuit outputs, the results predicted by the two models differed significantly, leading to over a 40 percent difference in the number of pin errors

predicted at the system level. The impact of these results is that logic-level injections may have to be restricted to gate inputs and outputs with only a single propagation path to circuit outputs.

CHAPTER 5.

COINCIDENTAL FAULT ANALYSIS

This section discusses an experimental approach for simulation-based validation of fault tolerant microprocessor architectures. The approach is intended to investigate critical aspects of such designs from a fault-tolerance viewpoint. The method is illustrated on an example of a fault-tolerant jet-engine controller (EEC131). In particular, the digital aspects of the dual-channel controller, described at the logic and functional levels, are simulated, and transient fault injections are performed. The coverage of the dual technique to single and correlated transients is evaluated.

In the simulated controller, fault detection and reconfiguration are performed through transactions over communication links. Instructions specifically designed to exercise this cross-channel communication are executed. The simulated fault-injection approach is illustrated by measuring the level of effectiveness of the dual configuration to transient errors. The results show that none of the single injections affects more than one channel, while approximately twelve percent of the multiple injections affect both channels; none results in controller failure since two additional levels of redundancy exist.

The next subsection contains the description of the experimental approach, and subsection 5.2 describes the experiment. Concluding remarks appear in subsection 5.3.

5.1. Coincidental Fault Injection Experiment

The focus of the simulation experiment was to stress the fault-tolerance mechanisms of the digital aspects of the dual system. The following aspects of the controller were simulated: both CPUs, at the gate and electrical levels to allow transient fault injections; the external modules, including I/O processors and memories (which were not subject to fault injection), at the functional level. Software to exercise the fault-tolerant operation of the dual system was programmed into the ROM and was executed by both processors. The executed instructions were intended to mimic the process of sampling of the engine oil temperature through the I/O processor, reading the sampled value from I/O processor, sending the value to the other channel through the crosstalk communication link, and receiving data from the other channel for comparison. The focus of the experiment was more on stressing the channel communications and less on attempting to use real data.

Transients with a charge level, between 0.5 pC and 8.0 pC were first injected into the microprocessor of channel A (for single fault injections) and then into both channels A and B (for multiple faults injections). The charge levels chosen represent the transient response of various heavy ions, including 100 MeV Fe ions, which are commonly found in the cosmic environment. These levels were chosen to ensure that no permanent errors occur. Charge levels approximately greater than 10 pC are known to cause permanent latch ups (device failures) [47]. Only the results for the charge level of 8 pC are presented here, since, for charge-levels between seven and 10 pC, the probability of error is

relatively constant, i.e., additional charge does not result in an increase in the error probability.

The locations of the fault injections were selected to maximize the chance of channel failures in the system. This method is similar to the *failure acceleration* technique proposed in [57], wherein it was shown that accelerated error-to-failure scenarios can be used to estimate fault impact. For example, a transient was injected directly to register R1 at a point in time when it contained critical input data. Thus, a worst-case situation for the controller was modeled in this experiment. Locations and time-points of the injections were selected to alter the data value stored in the accumulators and to stress the crosstalk communication between the channels. These I/O functions were the targets for the induced failures, since their short latencies allowed us to measure the error coverage without the overlapping effect of possible latent errors.

Transients were injected at eight selected nodes in the ALU and in the control units of the CPUs. Nodes were chosen that had low fanout, since the small capacitive loadings made them sensitive to logic upsets. Additionally, each node had fanouts to critical points in the CPU, e.g., the CPU registers, the CPU register control points and the external data-bus control lines. Time points of the injections were chosen to be the clock cycles before new values were latched into the registers. In all, over 80 fault injections/simulations were performed.

This number was determined by the fact that additional injection did not vary the result significantly.¹¹

Recall that a single-channel functional error is assumed to occur if the injected transient altered the critical input data in either R1 or R2. A dual channel functional error is defined as the event where the contents of CPU registers R1 and R2 are faulty in both channel A and channel B. The error data for the analyses were generated by comparing each faulted simulation with a fault-free simulation. The error data were then processed by a series of programs that collected statistics on the fault injections and the results.

5.2. Impact of Single and Multiple Fault Injections

The single-fault injection experiments resulted in three types of errors:

(1) *Error in channel A only:* In this case, critical data in register R1 in channel A became faulty, after the contents of R1 were sent to register R2 of channel B, i.e., a correct copy of the critical input data was sent to channel B. This was not a serious problem, since out of four registers R1 and R2 in channels A and B, only one (R1 of channel A) was corrupted. The system could sustain a second fault in one of the registers and continue to be operational.

(2) *Crosstalk error:* In this case, an error was introduced during the communication between the two channels in the dual system. During the process of sending the critical data from R1 in channel A, to R2 in channel B, the I/O processor

¹¹The determination of the number of fault injection also follows common statistical principles, since by the law of large numbers, a sample of greater than approximately 30 is expected to produce stable statistical distributions.

of channel A read faulty data into the I/O buffer from the bus. As a result, the I/O processor sent a faulty value to channel B. This was also not a serious problem, since as before three of the registers contained error-free data (only R2 in channel B is faulty). Here again the system could sustain a second fault with no apparent impact.

(3) *Error in both channels:* In this case, a transient altered the content of R1 in channel A before it was sent to channel B. Thus faulty data from R1 in channel A was sent to R2 in channel B, i.e., the data in both R1 in channel A and R2 in channel B were faulty. The system was still operational, because the data in R2 in channel A and in R1 in channel B were error free. However, unlike the above cases, a second fault could cause both channels to have functional errors, which is the most critical of the single fault conditions.

Table 5 summarizes the results of the experiment. In the table, the number of transient injections resulting in errors in each case is given. Note that approximately one in three transients causes an error in the critical data. Over eight percent of the injections result in alteration of critical data in R1 in only one channel. There is approximately a 12 percent chance that correct data from R1 in

Table 5: Error due to fault injections in the channel A.

Error Category	Error Frequencies	Percentage
No Error	51	63.8%
Error in Channel A Only	7	8.8%
Crosstalk Error	9	12.3%
Error in Channels A and B	13	16.3%
Total	29	36.3%

channel A is altered during the crosstalk communication to channel B. The probability of a transient causing an error in both channels is moderately high (16 percent). In each of the above cases, the controller continues to operate without failure because, one of the channels still provides the correct input data.

5.3. Multiple Fault Injection

The dual configuration of the system is quite effective in tolerating single event faults. However, in an actual operating environment, transients do not always occur in isolation. The chances of having multiple errors as a result of external current or voltage spikes or, as a result of transients occurring on the external input lines, may be significant. This is particularly so if the input lines are connected to multiple locations in the system. For example, a lightening strike on an aircraft can affect several sensor-input lines of the avionic equipment. The resulting errors may impact more than one CPU or component simultaneously and can alter critical input data in several registers. The impact of transients occurring at multiple locations, at the same time, is studied in this section. My fault injection methodology places particular emphasis on multiple errors that can result in altering critical input data in both channels. We do not, however, claim to accurately model the physical transients occurring in real avionic environments.

Table 6 shows the impact of the multiple transients in the target system. In the table, over 52 percent of the multiple errors result in functional errors in one channel. However, only 12 percent of the injected transients result in causing

Table 6: The multiple fault injection results.

Fault Category	Occurrences	Percentage
No Error	19	47.5%
Functional Error in One Channel	21	52.5%
Functional Errors in Both Channels	5	12.5%

functional errors in both channels, i.e., only one in every four functional errors affects both channels. The overall coverage of multiple transients is approximately 88 percent. The confidence interval for this estimate is calculated in the following section. Multiple faults that alter the contents of the R1 registers in both channels are seen to be critical since they will result in the invocation of the reserve value. An increase in the fault tolerance of the input data paths to the R1 registers and their control circuits may significantly improve this aspect of system dependability.

5.4. Confidence Limits

Assume that each multiple fault injection can cause both channels to have functional errors with a probability p . Assuming that the fault injections are independent, the probability of a functional error on each trial is p . It is easily seen that the random variable X (number of failures) has binomial probability distribution with $n = 40$, $p = p$ and $q = 1 - p$; \bar{X} in our experiment is 5. The estimated value, \bar{p} , from the experiment is 0.125 which equals the number of functional errors divided by the number of fault injections.

Since the number of trials is sufficiently large ($n = 40$), the probability density function of X can be approximated by a normal distribution with $\mu = np = 5.000$ and $\sigma^2 = np(1-p) = 4.375$.

We find the 90% confidence limit $[\alpha, \beta]$ for X as follows:

$$\text{Lower 45\% limit: } \alpha = \bar{X} - Z_{0.05}(\sigma/\sqrt{n})$$

$$\text{Upper 45\% limit: } \beta = \bar{X} + Z_{0.05}(\sigma/\sqrt{n})$$

Based on the above assumptions, the 90% confidence limits for the number of dual channel failures in the experiment is equal to $[4.456, 5.544]$, i.e., the coverage of multiple transients is $[86.1\%, 88.9\%]$ with 90% confidence.

5.5. Discussion

This section presented an experimental approach for simulation-based evaluation of a fault-tolerant feature in the target architecture. The approach used mixed electrical and logic simulations, combined with fault injections, to evaluate the susceptibility of fault tolerant designs to transient errors. The method was illustrated on the digital aspects of a fault tolerant, dual channel jet-engine controller. The coverage of the fault tolerance technique to single and multiple transients was evaluated. The locations and the time-points of the fault injections were selected so as to maximize the chance of channel errors. Specifically, faults were injected under conditions where critical communications were taking place within the dual system.

The results showed that the controller had an estimated 100 percent coverage against single isolated transients, while approximately 12 percent of the multiple transients affected both channels.

CHAPTER 6.

SOFTWARE UPSET ANALYSIS

This chapter describes a simulation based approach to quantify the impact of low-level transient errors at the software execution level. The experimental approach analyzes the software (program-flow level) upsets in VLSI systems. Automated fault-sensitivity analysis, for the runtime injection of transients at the device level and the assessment of the resulting impact on the program control flow, is proposed. Using test workloads, the type of upsets at the program-flow level which can result from fault propagation are determined. The mechanisms involved in internal propagation of latch errors and their effect on the software execution are modeled.

The approach is illustrated by a case study of the HS1602 microprocessor. For each section in the application program, the chances of having single and multiple upsets from the fault injection are determined. The analysis showed that about 20% of all upsets were multiple in nature. The results suggest that current methods of validation that assume single upsets may be inadequate. A determination of the error characteristics at the software level is performed and the faulty behavior of the software due to the hardware transients is quantified. A Markov model is constructed from empirical data to describe the error propagation within the microprocessor and the subsequent impact at the program flow. The model is used to identify the functional unit most sensitive to error propagation and the unit with the highest potential of causing software upsets.

6.1. The Experimental Approach

A number of techniques, at the hardware level, for fault sensitivity analysis have been proposed and implemented in [22]. They include transient impact assessment on latch, pin and functional errors, external pin error distributions due to within-chip transients, and error propagation models to depict the dynamic behavior of latch errors. The details of these analysis techniques are reported in the Chapter 3. An equally important issue that has to be addressed is the fault impact at the program control flow level. To quantify and model the fault impact on the executing software, we performed a combined analysis of the hardware and software behavior. In this section, error characterization methodologies and a software monitoring technique are developed to allow such combined analysis via simulation.

To characterize the software upset, faults are injected at runtime, at the transistor level, and their resulting error is propagated to gates and latches, through the chip's functional units. Resulting software upsets are detected through synchronous monitoring the hardware activity while the application program executed on the target design during the simulation.

A functional unit that is most likely to cause a given type of software upset is identified. The internal propagation of latch errors and their effect at software execution are modeled. The *faulty* hardware state (i.e., state of latch errors in the hardware for a given clock cycle) that has the highest chance of resulting in an upset, on the executed program, is isolated through modeling the latch error

propagation in hardware. Design feedback information is generated using such parameters. With such information, a designer can come up with a scheme to protect the functional unit most sensitive to error propagation and modify the design to avoid the critical *fault-situation* that can lead to a failure.

We use the mixed-simulation approach described in Section 3.1. The fault injection process is illustrated in Figure 22. The injected current source is specified as a mathematical functional and the resulting transients can be of varying shapes and duration. The user can control the location of a fault, the time and duration of a fault, and the shape of the current source. The details of validation of fault injection method appear in [22].

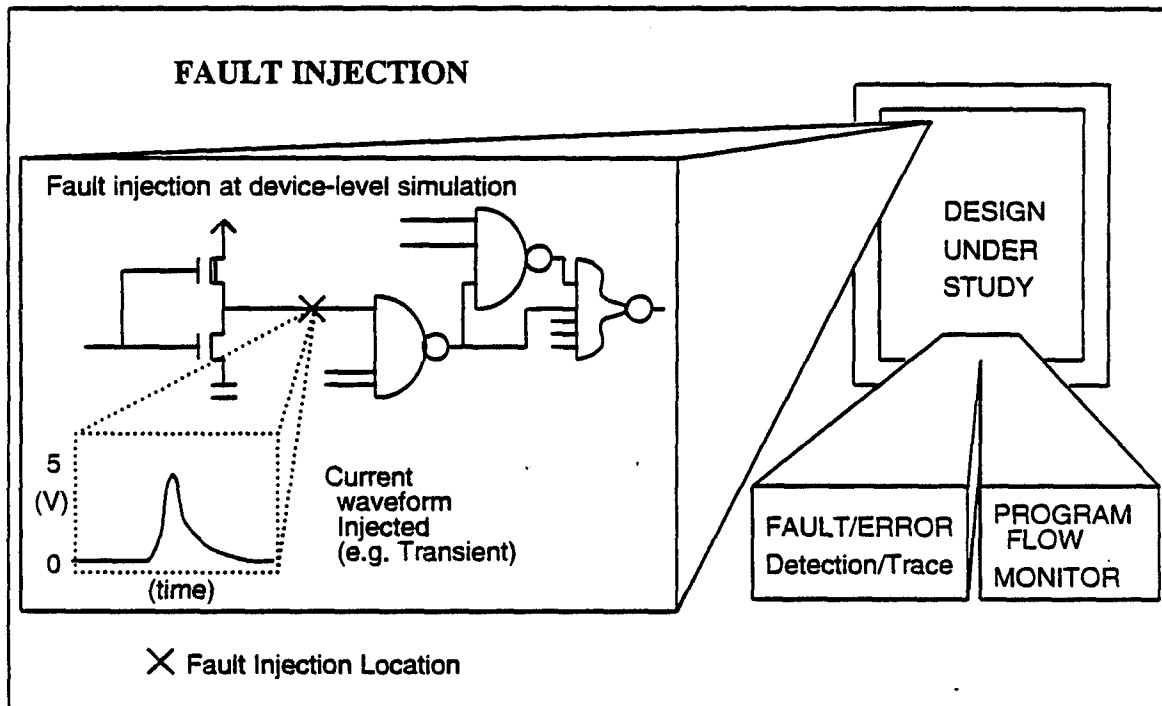


Figure 22: Fault injection illustration.

The *fault/error tracing facility* monitors all switching activities in the target system, including fault propagation through each gate or transistor, for all processed events. The trace data for each event consist of the time of the event, the hierarchical node name, and the new and previous voltage levels (for electrical nodes), or the new and previous logic levels and their strengths (for logic nodes). The *program-flow monitoring facility* captures the program execution information in the target system during the fault injection and subsequent error propagations. The executed program sequence and processed data are traced through monitoring at counter register, program-address bus, data registers and data bus. With the program-flow trace, upsets occurring at the software level can be detected.

6.2. Fault/Error/Upset Analysis

Recall that the tracing facility is capable of monitoring each node. The error data for the analysis are generated by comparing each faulted simulation with a fault-free simulation. An error is assumed to occur if the injected transient causes the node voltage to vary beyond a defined logic threshold. For each simulation, the recorded data include the time of fault occurrence, the location of fault, the faulted value, and the fault-free value. Each fault event is also classified as either a timing error (premature or late firing) or a value error.

Statistics are collected on the fault injections that result in latch errors and upsets (altering the flow of an application program). An error is assumed to occur when an erroneous value is detected at the output of a latch or monitored

node, and an upset is assumed to occur when the program counter or a data register value is altered.

System error state is characterized in the following way. From the collected data, the system *error-signature* is generated by counting the number of latch errors by functional units, for each clock cycle following the fault injection. Figure 23 illustrates capturing such error-states. For example, during the clock cycle t_i there are 2 ALU latch errors and 1 control unit(CNT) latch error and so on.

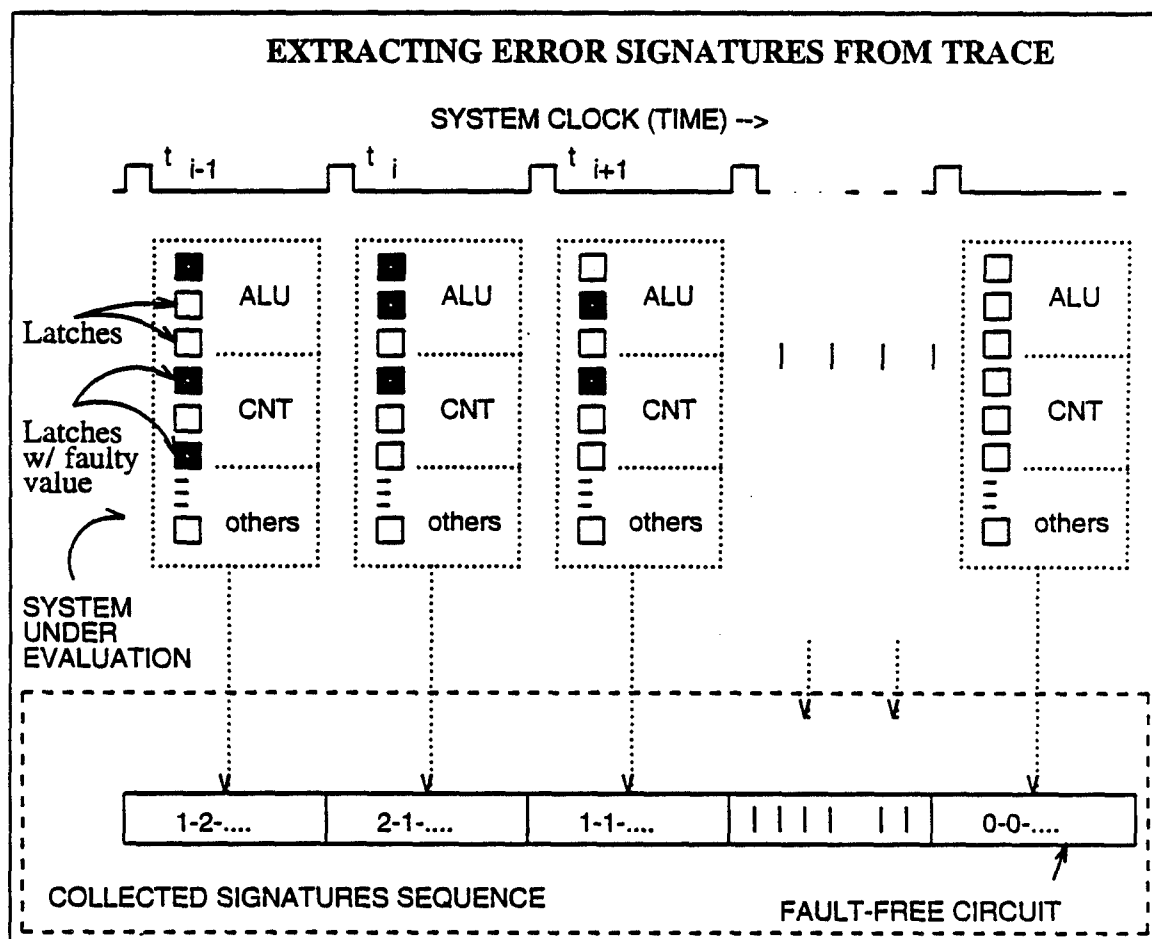


Figure 23: Error-signature generation.

The signature for t_i is then given by 2-1-... (ALU-CNT-...). A sequence in time $(t_0, t_1, \dots, t_i, t_{i+1}, \dots)$ of error signatures is generated for each fault injection until the system becomes fault free or fails completely (defined by a processor stall). For each clock cycle, the *faulty state* of the system is represented by its error signature. Thus the change in error signature from one clock cycle to the next clock cycle represents the system's response to the internal fault propagation.

All observed error signatures from the fault injection experiment are clustered into *representative* states using a statistical clustering algorithm based on Euclidean distance. A Markov transition model to depict the dynamics of the space and time fault propagations within the chip is then generated with transition probabilities calculated from the sequence in which error-signatures occur. The probabilities of transition between the *representative* error-signature states and detected program upsets are quantified in the Markov model.

A single or multiple program-flow level upset is detected when the injected transient becomes activated and propagates to alter a value in the program counter, the program-address bus, data registers or the data bus. We define a single event program upset as a situation when an injected transient causes a single point flip of a logic value in the monitored location in time and space. Similarly, a multiple event upset occurs when a transient propagates to more than one monitored location in time and space.

6.3. Upset Classification

The analysis environment quantifies the fault sensitivity of a chip by evaluating a number of measures. The probabilities of latch error, data error and program deviation are calculated. Analysis of the error/upset data is performed to determine the effect of the injected transients on the severity of latch errors and the impact on the flow of the executed application program. Upsets are categorized into classes according to their incorrect program-flow scenario. Next, the Markov model to depict error propagation within the chip and the software-level program execution is generated. The following terminology is used in the analysis.

- 1.) Latch Errors: Fault injections which result in voltage transients that cause errors in latch outputs.
- 2.) Error Signature: Distribution of latch errors in functional unit of the chip for a given clock cycle.
- 3.) Data error: Fault injections which result in voltage transients that cause errors in any of the processed data by a studied system.
- 4.) Program Deviation: Fault injections which result in altering the flow of the executed program in the studied system, e.g., a transient can alter address bus while loading in a target address for a jump instruction.

- 5.) Single event upset: Fault injection which results in a data error or a program deviation at exactly one point, in time and space.
- 6.) Multiple event upset: Fault injection which results in more than one data upset and/or program deviation.

Program-level upsets are categorized in Figure 24. Single and multiple data error and program deviations are illustrated. The dotted line shows the *correct* program-flow path and the solid line shows the *observed* program-flow path. Any combination of data error and/or program deviation can occur from fault injection. The detailed approach of the analysis is illustrated via the case study in Section 6.5.

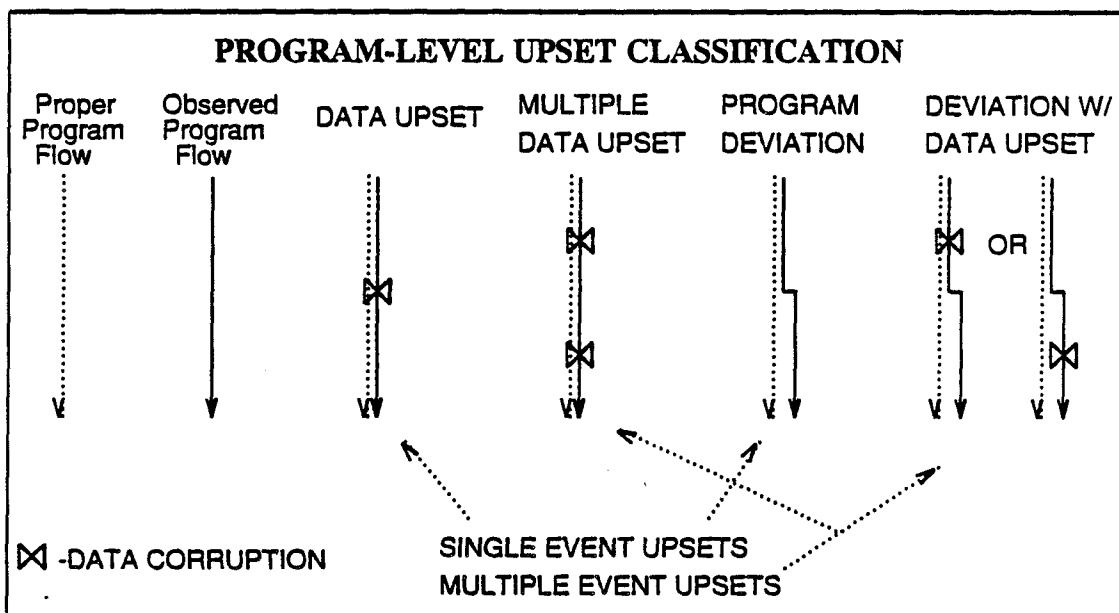


Figure 24: Program-flow level upsets.

6.4. Fault Simulation

Several issues are addressed in the simulations. Emphasis is on executing and modeling of a realistic workload under varying situations of the system under study. Application codes were executed on the target system during each simulation period (Figure 25).

The suite of application codes was carefully selected to ensure that different aspects of the microprocessor functions were exercised. The codes consist of four software sections. Each section exercises a different aspect of the design under study. Section A exercises timer circuit. It contains the code that generates synchronizing signals and a reset signal in case of parity error or a failure of the software sanity timer. In section B, the arithmetic and logic functions are

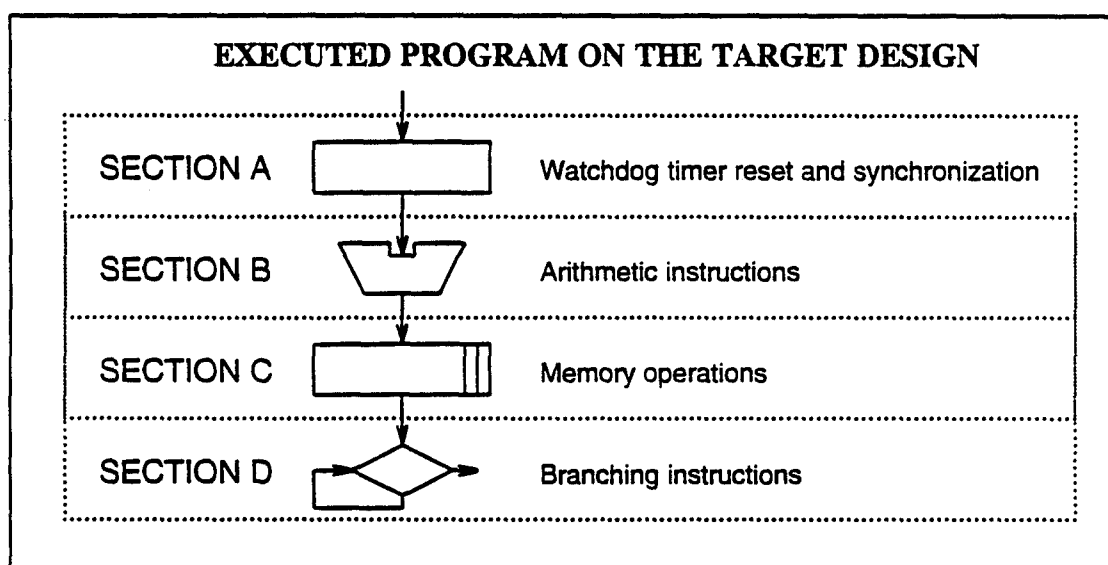


Figure 25: The application program.

exercised and section C performs different memory operations. Section D contains branching instructions.

Transient faults were injected during the execution of each program section and the program flow in the chip was monitored. The error signatures were also collected from every latch in the system for each clock cycle. About 40 instruction cycles (45150 nsec) of the application code were executed on the target system for each program section. Simulation resolution was 1 nsec or less. Each fault-injection simulation run took on the average about 20 minutes on a SUN IPC-SPARC workstation. A total of 1400 simulations were performed. During the simulation all the nodes (including all latches and external I/O pins) in the circuit were monitored and processed.

To establish simulation accuracy, a fault-free simulation run was compared with the operation of the actual hardware unit which is currently in operation at the NASA Langley Research Center. Two comparisons were made. The first verifies the correct flow and execution of instructions by monitoring data, address, and control lines during the times that these lines are stable (logically valid). Correct execution flow was observed for the entire simulation period. The second comparison, which is more rigorous, monitors all changes in logical values, including transition times. This comparison reveals electrical characteristics of the simulation, such as propagation delay, race conditions and gate loading. The result showed that the changes in the logical values for both the hardware unit and the simulated model are identical, but the times of the transitions are slightly skewed. The timing skew was due to the variance of several

design parameters. Gate delay, for example, is specified as minimum, typical, and maximum delay time. A normalization was needed because the actual system runs at 12.08 MHz and the simulation was set at 82 nsec per clock cycle (12.195122 MHz).

6.5. Results

This section provides a detailed illustration of the analysis approach via the actual results for the target system. The impact of injected faults at the latch and software level is determined. Also an empirical model of fault manifestation is derived from the data. These results provide an overview of the fault propagation within the chip and as such give a general evaluation of upset sensitivity of the design.

6.5.1. Upset severity

Figure 26 summarizes the overall impact of injected faults. In the figure, number (and probability) of total number of fault injections, latch error probability and probabilities for single and multiple upsets, during the execution of each program section, are shown. The first row shows the number of fault injections and the second row shows the number of those that result in latch errors. The third through sixth rows show evidence of having data upset, multiple data upset, program deviation and program deviation with data upsets, respectively. The last row shows the combined upset probability for each program section.

PROGRAM-FLOW LEVEL IMPACT OF INJECTED FAULTS					
Upset Category \ Program Section	PROG A	PROG B	PROG C	PROG D	ALL PROG. COMBINED
Number of Fault Injections	350 (100.0%) (25.0%)	350 (100.0%) (25.0%)	350 (100.0%) (25.0%)	350 (100.0%) (25.0%)	1400 (100.0%) (100.0%)
Number of Latch errors	84 (24.0%) (16.4%)	183 (52.3%) (35.7%)	149 (42.6%) (29.1%)	96 (27.4%) (18.8%)	512 (36.6%) (100.0%)
DATA UPSET	32 (9.1%) (16.9%)	74 (21.1%) (39.2%)	45 (12.9%) (23.8%)	38 (10.9%) (20.1%)	189 (13.5%) (100.0%)
MULTIPLE DATA UPSET	0 (0.0%) (0.0%)	13 (3.7%) (61.9%)	8 (2.3%) (38.1%)	0 (0.0%) (0.0%)	21 (1.5%) (100.0%)
PROGRAM DEVIATION	3 (0.9%) (5.6%)	22 (6.3%) (40.7%)	5 (1.4%) (9.3%)	24 (9.6%) (44.4%)	54 (3.9%) (100.0%)
DEVIATION W/ DATA UPSET	2 (0.6%) (5.7%)	12 (3.4%) (34.3%)	21 (6.0%) (60.0%)	0 (0.0%) (0.0%)	35 (2.5%) (100.0%)
TOTAL UPSETS	37 (10.6%) (12.4%)	121 (34.6%) (40.5%)	79 (22.6%) (26.4%)	62 (17.2%) (20.7%)	299 (21.4%) (100.0%)

Figure 26: Impact of injected faults.

Note that faults injected during the execution of program B had the highest chance of having upset. Forty and one-half percent of all observed upsets were contributed by program B. In general, ALU operations (program B) are most susceptible to upsets. Program A had the lowest probability of having upsets. Only 37 out of 84 latch errors resulted in upsets, because there are many logic rewrites in the timer reset operations.

The last column in the figure shows the combined result for all programs (A-D). Note that there is very high probability (13.5 percent) of having single

data upset. Such single upsets can be easily detected and corrected by the error-correction scheme. Overall, there was a 21.4 percent chance of an upset occurring. Multiple upsets (multiple data and deviation w/ data upset) accounted for about 20 percent (51 out of 299) of all upsets. Note that multiple upsets are not recoverable with recovery schemes often implemented, e.g., single bit parity correction circuit will fail under multiple bit flips, and a rollback recovery scheme will not function under program deviation with corrupted rollback variables. Only programs B and C resulted in multiple data upsets.

6.5.2. Error propagation modeling

Data were collected on the fault injections that resulted in a voltage transient large enough to produce latch errors and upsets. From the collected data, system *error-signature* sequence was generated. All observed error signatures from the fault injection experiment were clustered into *representative* states to generate a discrete time Markov transition model. Figure 27 shows a Markov model based on the measured data, to quantify the in-chip latch-error propagations. In the figure, the value of a state is the *representative* error signature for a clock cycle in question. Thus, given a latch-error condition of the system, the model shows the probabilities of staying in the current state, going into a different state or becoming fault free. For example, the probability of a fault injection resulting in STATE 1 is 0.14. At STATE 1, the probabilities of staying in STATE 1 is 0.04 and the probability of becoming fault free is 0.84.

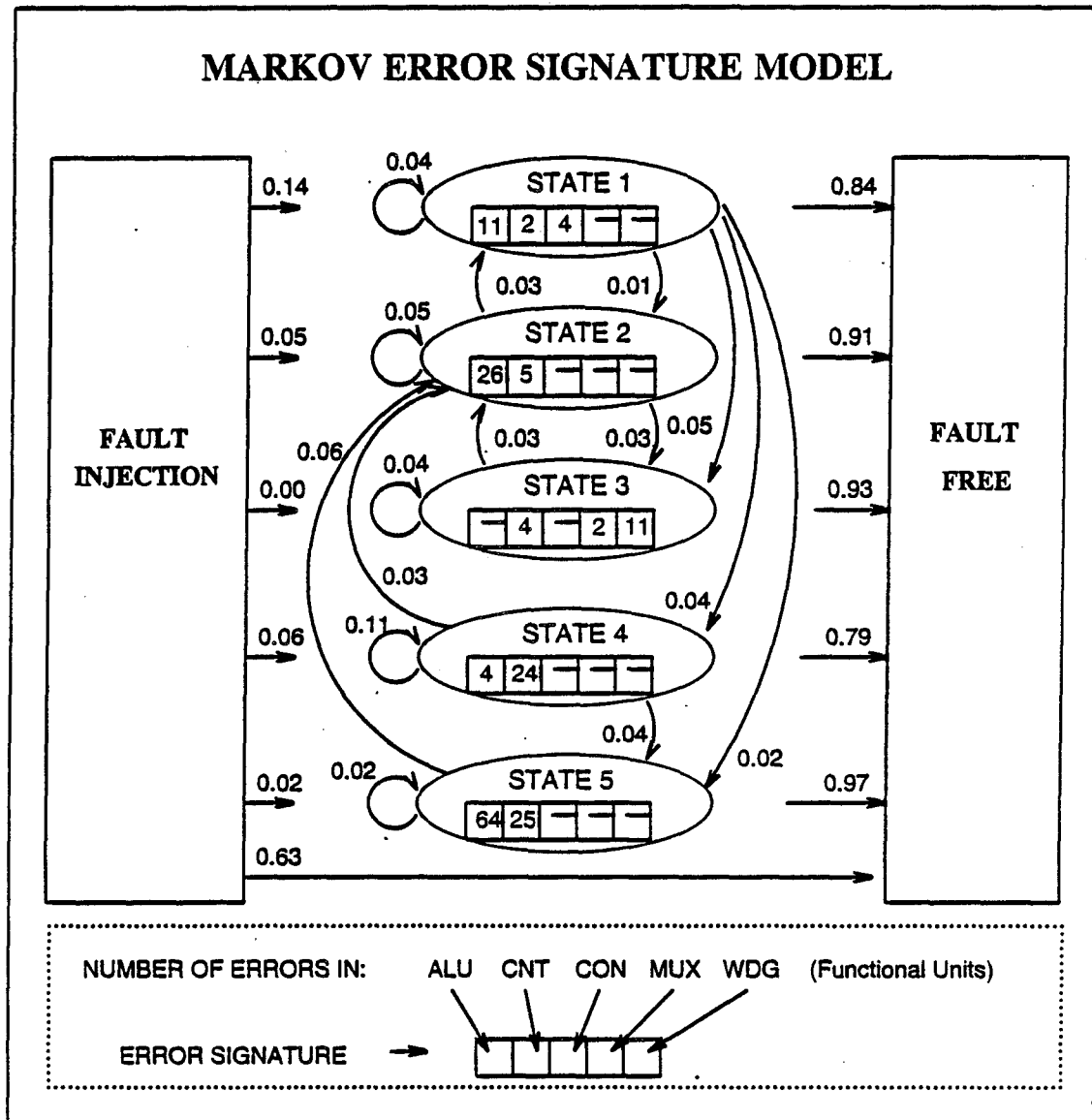


Figure 27: Error state transition model.

From the model, we observe that the latches in the ALU, followed by the latches in the control unit, are most susceptible to error propagation. STATE 5 has the highest number of error count, i.e., given that system goes into STATE 5, on the average, there will be 64 latch error counts in the ALU unit and 25 in the

control unit. It is clear that these two units are the most likely candidates for error masking (e.g., redundancy).

Figure 28 shows, for each state, the probability of having an upset. Using this figure the severity of different states can be compared. For example, from Figure 27, STATE 5 and STATE 2 have somewhat similar error signatures. However, as seen in Figure 28, STATE 5 and STATE 2 represent two distinct faulty states. STATE 5 has a high probability of causing single data upset, while STATE 2 has a high probability of causing program deviations and multiple data upsets. Thus, of these two, STATE 2 is easily the more *critical* (multiple upsets).




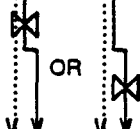
ERROR STATE IMPACT ON SOFTWARE					
Given an error state in the Markov model, what is the chance of having one of the following program upsets?	DATA UPSET	MULTIPLE DATA UPSET	DEVIATION PROGRAM	DEVIATION WITH DATA UPSET	TOTAL UPSETS BY STATE
					
STATE 1	12 (46.2%)	0 (0.0%)	14 (53.8%)	0 (0.0%)	26 (100.0%)
STATE 2	57 (52.3%)	15 (13.8%)	28 (25.7%)	9 (8.3%)	109 (100.0%)
STATE 3	24 (100.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	24 (100.0%)
STATE 4	23 (41.1%)	0 (0.0%)	7 (12.5%)	26 (46.4%)	56 (100.0%)
STATE 5	73 (86.9%)	6 (7.1%)	5 (6.0%)	0 (0.0%)	84 (100.0%)

Figure 28: Software upset result.

When latch error occurs in the multiplex or the watchdog units, the system must be in STATE 3, since (from Figure 27) this is the only state that has latch errors in the two functional units. Latch errors in the multiplier and watchdog units are not so crucial since STATE 3 can only result in single data upset.

6.6. Discussion

This section presented the method to quantify the impact of low-level transients at the software execution level. The software (program-flow level) upsets in VLSI systems have been analyzed using the experimental approach. Automated fault-sensitivity analysis, for the runtime injection of transients at the device level and the assessment of the resulting impact at the program-flow, was performed. Using test workloads, the types of upsets at the program-flow level which can result from fault propagation were determined. The mechanisms involved in internal propagation of latch errors and their effect at the software execution were modeled.

The approach was illustrated by a case study of an HS1602 microprocessor. Four sections of the application code were executed during the analysis. For each section in the application program, the probabilities of having single and multiple upsets from the fault injection were determined.

A Markov model was constructed from empirical data to describe the error propagation within the microprocessor and the subsequent impact at the program flow. The model is used to identify the functional unit most sensitive to error propagation and the unit with the highest potential of causing software upsets.

Different error characteristics at the software level have been determined.

Overall, there was a 21.4 percent chance of having an upset. About 20 percent of all observed upsets were multiple in nature. Arithmetic operations in the application program are most susceptible to upsets. About 40 percent of all observed upsets were contributed by program section B, which consist of arithmetic instructions.

From the generated empirical model, we observe that the latches in ALU are most susceptible to latch error propagations, followed by the control unit. STATE 5 has the highest number of error counts. If partial redundancy is the employed design approach, the protection of ALU and control units would be most beneficial since ALU and control latches are must susceptible.

CHAPTER 7.

PERMANENT-FAULT ANALYSIS

This section describes a simulation based approach for predicting permanent faults in VLSI designs. The approach combines the switch-level circuit simulation and the device-level Monte Carlo simulation to achieve realistic reliability assessment. A system under investigation is first simulated at the switch level and trace data on switching activity are collected. This trace data are then used along with Monte Carlo simulation to model wear-out at the device level, due to different failure mechanisms. The approach currently supports two failure mechanisms: electromigration and gate oxide breakdown. The Monte Carlo analysis uses *importance sampling* to reduce the run lengths. The key advantage of this approach is that it can closely mimic dynamic sequences of events in a physical device through time for the specified failure mechanisms. The technique can localize the weak location/aspect of the target chip and generate the TTF distribution of a VLSI chip as whole based on traces from circuit simulation using actual application codes and under realistic operating condition.

The process of electromigration is modeled by removing elements (metal grains), based on a probability calculated from normal charge distribution, in a matrix that depicts a metal line. This process is carried out in parallel for all the metal lines in the circuit. A metal line failure, i.e., chip failure is assumed to occur if a void path is created from the left to the right edge of the matrix. The analysis is performed a number of times to determine the distribution of the TTF

of the target chip. Concurrently, a similar Monte Carlo approach is used to simulate failures due to dielectric breakdown.

The use of this approach for evaluating the reliability of a design is illustrated with a case study of the HS1602 microprocessor. The simulation analysis is performed under varying operating environments and fabrication technology parameters. In particular, operating voltage, temperature and device dimension are varied and the reliability impacts of reduced dimension and technology improvements are quantified. The method is illustrated by applying it to predict the TTF characteristics of a microprocessor chip in a typical operating environment (room temperature, 5 V).

The next subsection describes the experimental approach, and the experimental analysis of the target system is illustrated in subsection 7.2. Concluding remarks are in subsection 7.3.

7.1. The Experimental Analysis Approach

The approach combines circuit simulation and Monte Carlo analysis to simulate the wear-out processes for an entire IC chip. Figure 29 depicts the overall experimental approach. First, accurate simulation of the target chip is performed to acquire trace data on switch activity, using a hierarchical switch-level simulator, SPLICE and running actual application codes. A tracing facility monitors switching activities on all of the internal nodes. Then, using the switching activity information (trace data), the electromigration wear-out and the oxide breakdown processes are simulated using Monte Carlo techniques. For example,

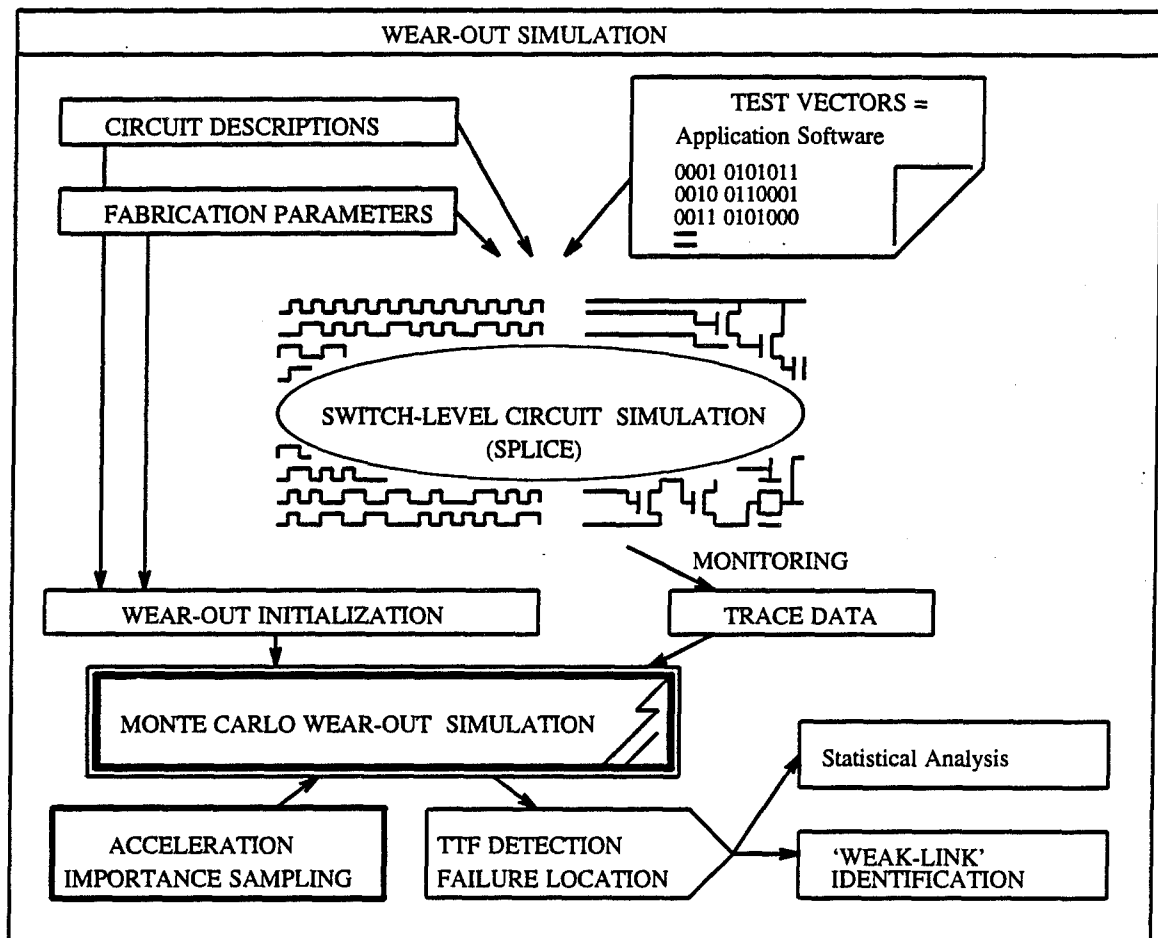


Figure 29: Experimental approach.

to perform Monte Carlo simulation of electromigration, a failure site on a metal line is modeled by a matrix of metal grains, and grains are removed based on a probability calculated from the empirical data. The grain-removal process is performed until a line failure has occurred, i.e., a void has formed on a line. The importance sampling method is used to accelerate the wear-out process. The resulting TTF data are collected, and locations of the 'first-to-fail' nodes are identified.

Other proposed approaches for reliability prediction mainly consist of deriving analytical formulas from plausible physical assumptions, with computational results that can be tested against available empirical data. Such methods provide closed-form information such as time-to-failure at component level, but they lack the ability to generate important information such as accurate distribution of time-to-failure of a VLSI chip as whole, which a simple MTTF calculation cannot provide. My goals were to overcome these shortcomings of conventional approaches and to develop a realistic model of the wear-out related failures that occur during the operational phase for a system.

Advantages:

The key advantages of the Monte Carlo approach over other proposed methods are:

- I. Fail-sensitive locations in the circuit can be identified at the design phase, and effective feed-back information for designers can be generated. Also, the Monte Carlo technique allows the user to observe dynamic processes of wear-out in the target chip. Such observation of different wear-out processes can be valuable information at the design stage.
- II. The Monte Carlo approach can effectively model the workload-reliability dependency of VLSI systems. In order to obtain useful predictions, reliability analysis based on realistic workload is important. In our experimental approach

a target system can be simulated with its actual application code to generate traces for Monte Carlo wear-out analysis.

Section 7.1.1 describes the simulation of the target chip to acquire trace data on switch activity; Section 7.1.2 develops the simulation of electromigration and oxide-breakdown via Monte Carlo techniques. The field device failure, due to wear-out, occurs in the order of years. Section 7.1.3 describes an acceleration technique that allows simulation of such long-term effect.

7.1.1. Logic simulation

A hierarchical timing simulation at the switch-level using SPLICE [45]¹² is performed. About 80 instruction cycles (90,300 nsec) of the actual application code were executed on the target system during the simulation period. Simulation resolution was 1 nsec or less. The suite of application codes was carefully selected to ensure that all of the functional units were exercised.

To obtain a comprehensive switching activity in the microprocessor, a tracing facility was also developed to monitor all of the internal nodes of the target chip. The tracing facility is capable of monitoring each node for all processed switching events. Figure 30 illustrates the circuit-level workload generated from the simulation. For each node in the circuit, the voltage level is monitored and events are created when the voltage levels pass a logical threshold. An event is either $e1[t]$ (change from logic level 0 to 1 at time t) or $e0[t]$ accordingly. The

¹²The switch-level analysis in SPLICE was performed using a relaxation based method that uses MOS oriented models. Virtually unlimited levels of signal strength can be associated with each of the logic values in order to further enhance the accuracy. This approach allows a correspondence between the electrical output conductance and the logic output strength. A fanout-dependent delay-model capable of handling first-order effects is used to achieve accurate delay-handling.

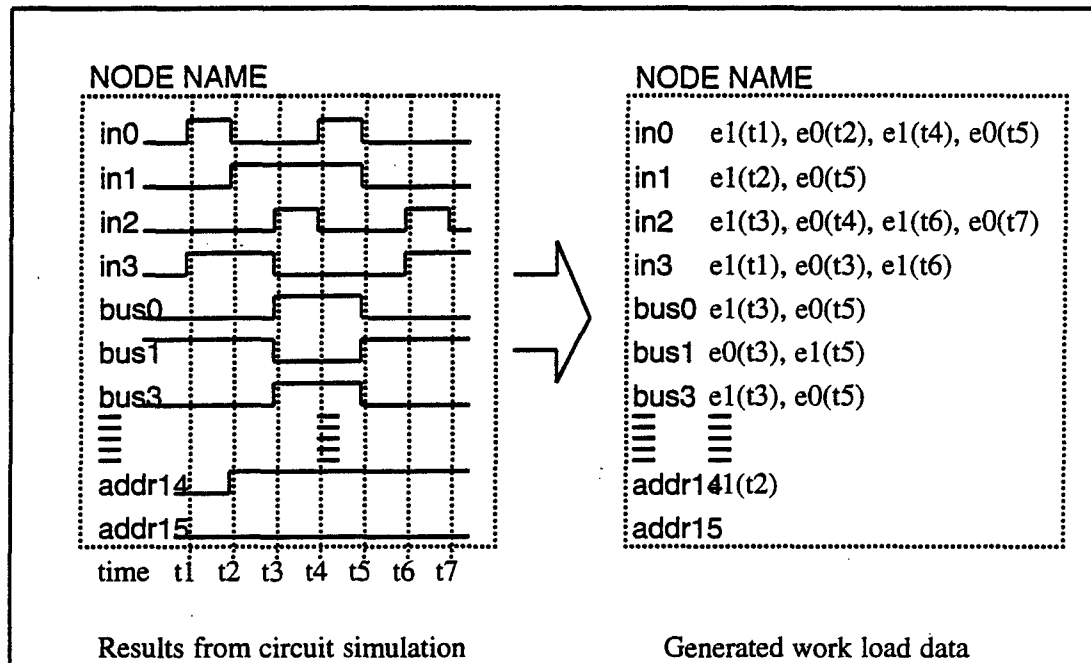


Figure 30: Work load data.

trace data for each event consist of the time of the event, the hierarchical node name and the new and previous logic levels and their strengths. The trace data from the simulation are used as implied workload on the target system during the operational period. The trace data are used along with the Monte Carlo simulation to model wear-out at the device level, due to electromigration and oxide-breakdown as described in Section 7.1.2.

7.1.2. Monte Carlo simulation

7.1.2(a). Electromigration

The process of electromigration is modeled by removing elements (metal grains) in a matrix that depicts a metal line. This approach attempts to mimic the

actual wear-out processes as given in the graphical metal-layer model used in [38]. A more elaborate model such as the two-dimensional model proposed in [38] can be used for higher accuracy but it would not be suitable for analyzing a whole chip. My simplified model allows reasonable computational time and storage space requirements for analyzing thousands of nodes simultaneously (for each metal interconnect in a VLSI circuit).

The depicted matrix approximates the *critical region* near the switching device in the metal line where the chance of having electromigration is the highest.

Size of the matrix: The size of the matrix is chosen to mimic the actual grain size in the metal layer. The size of the matrix is $(n \times n)$, where n is equal to the number of grains across the width of the metal line. Thus, the relative size of grain to the width of a metal line is reflected on the size used for the matrix. The size of the matrix determines the resolution (how roughly the failure steps will take place) of failure process. This approach approximates the actual grain dislocation processes due to the current stress.

An example of a metal line model is shown in Figure 31. The line shown in the figure has the actual device dimension of the target chip used in our case study. The width of the lines is 3.3μ and the grain size (diameter) of 1.1μ . The metal lines in the chip are approximately 0.25μ thick.

Initialization: Initially, N matrices of the size $(n \times n)$ are considered to model all of the N nodes in the circuit. The nodes that have a very low switching rate (having low failure probability) are dropped and 350 most likely-to-fail nodes are

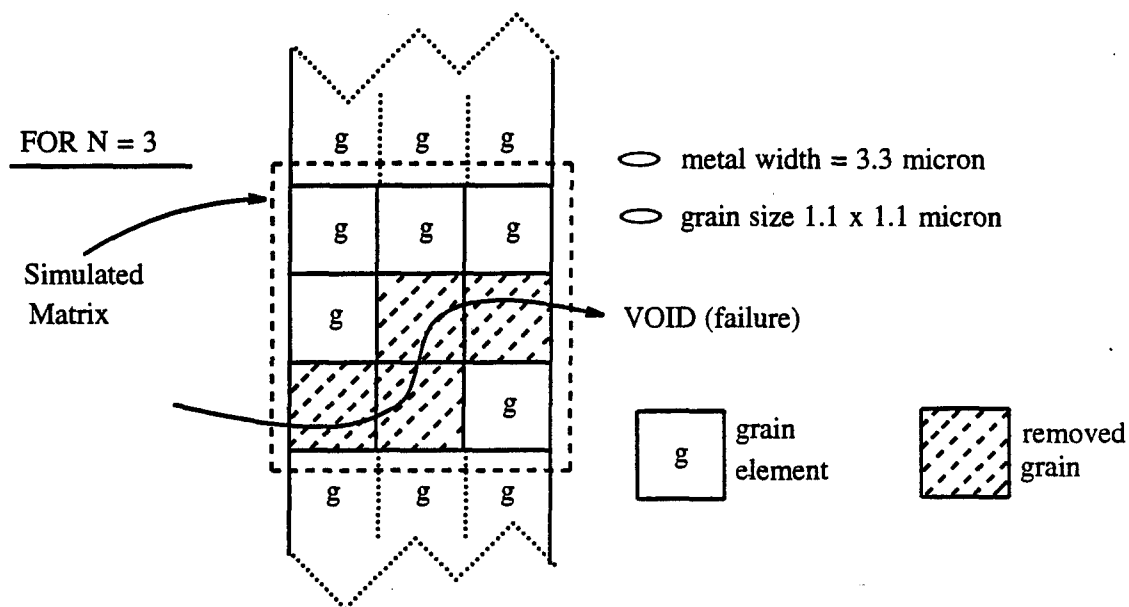


Figure 31: Modeled metal line.

considered for the analysis. The number of nodes to be considered was chosen such that time and space requirements for an exhaustive analysis is reasonable. Once matrices are initialized, elements are removed based on probabilities calculated in the subsequent steps upon detection of an event, from the circuit simulation trace, associated with the matrix (node) in question.

Wear-out modeling: Once the matrix size is selected and initialized, we begin the Monte Carlo process by reading and processing trace data from circuit simulation. Each switching event from circuit simulation triggers a Monte Carlo evaluation of the matrix that depicts the gate/node associated with the event. The trace from circuit simulation is repeatedly applied, and the grain removal process continues until a failure occurs.

Probability of grain removal calculation: To calculate the probability of a grain removal during each switching event, we first determine the amount of charge flow, through a grain, needed to fail (remove) the grain. The mean and variance of this variable are calculated from empirical data. Then we calculate how much current is actually imposed on the boundaries of the grain during each switching. Using the distribution of charge-amount to fail-probability, we calculate the probability of failure for each switching event. The specific steps in the foregoing calculations are as follows:

STEP A: Using empirical TTF data from physical experiments, we determine the distribution of the charge amount that must be exerted on a given line/grain to cause an electromigration failure.

- i. First, we obtain the mean and variance (in *seconds*) of the TTF distribution from reported experimental data.¹³ Typically, the empirical TTF distribution is log-normal. For simplicity, we assume that it is normally distributed. Since the variance-mean ratio of the distribution is very small, this assumption is reasonable.
- ii. Next, the average current I_o (in *amperes*) that was used in the physical experiments is also obtained from experimental data. From the average current for the device tested, current I (in *amperes/cm²*) per-unit-area is calculated by dividing I_o by the dimension of the cross section (in square centimeter) of the metal line.

¹³Many authors report experimental results on the determination of TTF of metal lines in ICs through physical testing. We use the results reported in [36] since they appear to be representative for the technology similar to that we are concerned with.

$$I = I_0 \sqrt{\text{cross_section_of_tested_metal_line}} \quad (1)$$

iii. We transform the TTF distribution to charge-to-failure (in C) distribution by multiplying the mean and variance of the TTF distribution by the average current calculated from Equation (1).

$$\text{Charge_to_failure} = \text{TTF} \times \text{Current}(I) \quad (2)$$

iv. We scale the distribution obtained in Equation (2) by the the cross section area of the metal grain used in our design. We perform this scaling by the following equation:

$$\begin{aligned} &\text{Charge_for_failure}(\text{scaled for grain removal}) \\ &= \text{Charge_to_failure} \times \text{Grain_cross-sectional_area} \end{aligned} \quad (3)$$

Here, we are assuming that the amount of current density needed to fail a given metal line varies linearly with respect to cross-sectional area. This assumption is reasonable because the atomic flux¹⁴ due to the field generated from a given electronic current density is proportional to the cross-sectional area (assuming that other device parameters such as temperature, conductance and activation-energy remain unchanged). Once we obtain the amount of current density required to remove a grain, the next step is to determine the current density in through the target metal line during each switching event. In the following step, we calculate current through a node during each switching event.

¹⁴Migration of metal atoms through a confined region in space. In our case, cross section of a metal line is the confinement.

STEP B: For each switching event, charge that would be exerted on the metal line is calculated from the metal length, floating capacitances and operating voltage. The amount of charge that flows through the *critical region* (which is modeled with a matrix) of a metal line, in a switching event, is the total charge that the metal line can hold. The integral of the voltage-capacitance product is the total charge that the metal line can hold:

$$\text{Charge_Due_to_Switching} = \int \text{Capacitance (F)} \times \text{Operating Voltage (V)} dt \quad (4)$$

Then, to find the charge amount that goes through a single grain, we multiply the charge due to switching from Equation (4) by the ratio of the cross-sectional area of a grain and that of the metal line:

$$\begin{aligned} &\text{Charge_Due_to_Switching (for grain)} \\ &= \text{Charge_Due_to_Switching} \times \frac{(\text{cross_section_area_of_grain})}{(\text{cross_section_area_of_line})} \end{aligned} \quad (5)$$

STEP C: Using the result from Equation (5), we calculate the grain removal probability from the charge-to-failure distribution generated in step A as illustrated in Figure 32. The probability of grain removal for a switching event is the area under the probability distribution up to the charge value calculated in Equation (5). The following example illustrates the grain-removal probability calculation:

STEP A: The mean TTF reported in [36], for the average current ($2\text{E}+6 \text{ A/cm}^2$ [normalized] for tested wire) applied to the tested lines, is 774,000 sec [$\sigma=3096$]. The charge value of $1.55\text{E}+12 \text{ C/cm}^2$ (variance: $\sigma=6.19\text{E}+9$) is obtained by

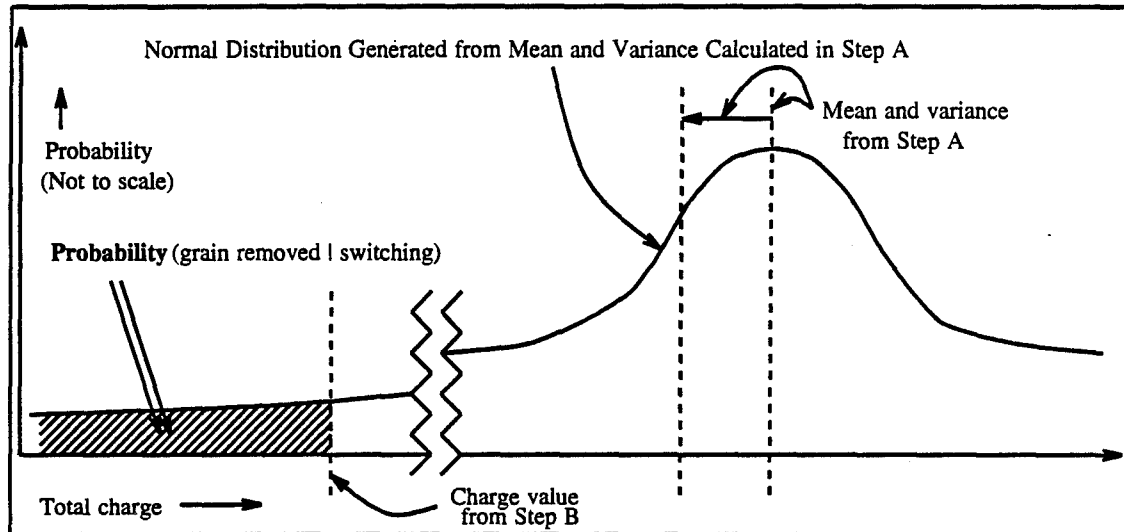


Figure 32: Grain removal probability calculation.

multiplying the mean TTF by the average current. This is the mean value of the charge-to-fail distribution for the tested line. Scaling this distribution to a $1.1 \mu \times 0.25 \mu$ wire section (for a *grain* in our target system design), we have mean charge of $5.64\text{E}+4 \text{ C}$ (Normal distribution: $\sigma = 2.24\text{E}+0$) for the charge-to-fail distribution for a grain removal in our simulation.

STEP B: The charge that flows during a switching event, through the *critical region*, is calculated in this step. For the target chip used in our experiment, a $3 \mu \times 1520 \mu$ wire ($8.73\text{E}-4\text{F}/\mu^2$), for example, has a net capacitance of $3.98\text{E}-12\text{F}$ (3.98 pF). Thus, in each switching event, at the operating voltage of 5 V, the total charge that flows through the critical region in the metal line is 19.9 pC. Scaling this result, to the dimension of a grain, gives us the following

result: 4.98 pC for a grain ($1\ \mu \times .25\ \mu$ section). We perform a similar calculation for every line considered for simulation based on layout information.

STEP C: In this step we combine the result of steps A and B to generate the grain removal probability for each line for a switching event. From the normal distribution of charges for a grain removal calculated in step A, $N(564E+4, 2.24E+0)$, we approximate the probability of grain removal $P(x < 4.98)$ to be about $2.3E-5$. Thus, the probability of a grain removal, for the example metal line, in each switching event, is 0.000023.

Monte Carlo process: The above steps described how to obtain the grain-removal probability in each switching event. This section discusses the Monte Carlo evaluation of the electromigration wear-out mechanism. Initially, a clean matrix (all entries initialized as 1) is assigned to each node, and the grain-removal probability for each matrix (node), given a switching is calculated and stored in a look-up table. The grain-removal probabilities for all grains in a same line are the same. Once initialization is complete, the simulator iterates, in time, until a failure is detected. In each time-iteration step, if an event is found in trace data, the wear-out mechanism is evaluated for the matrix corresponding to the node at which switching has occurred.

For each grain in the evaluated matrix, a random variable (assigned with a random number generated from the uniform distribution $[0.0, 1.0]$) is compared with the grain-removal probability found in the look-up table. If the random variable falls within the 'fail-region' of the grain-removal probability space, then

the grain is removed from the matrix. Thus, at any given time, an entry in a matrix can be voided (0) (i.e., grain has been removed) or intact (1). The state of a matrix is the condition of entries in the matrix, e.g., (011,111,111) is a state in which the first row - first column entry is void and the rest are intact. Once a grain is removed, the removal probabilities for the grains in the same column of the matrix increase appropriately to compensate for increased current density now through a thinner metal region. A graphical example of the grain-removal processes and metal line failure is illustrated in Figure 33. The above process continues until a failure is detected. Trace data from circuit simulation are repeatedly applied. Figure 34 describes the electromigration-simulation algorithm.

7.1.2(b). Oxide breakdown

This section describes the Monte Carlo approach to simulating the effect of long-term circuit operation on gate-oxide reliability. It is often assumed that the overall reliability of oxides in a chip is limited to the given fabrication technology, which includes the quality of the silicon material, the cleanliness of the fabrication environment, and the gettering and oxidation conditions. The key concerns for most of the work in this area have been defect-related failures. Every weak spot, i.e., defect point, behaves as a normal oxide with a thickness that is smaller than the nominal thickness. Simple and fast capacitor breakdown tests [40] or other conventional testing/diagnosis can determine the failing spots related to defect. However, not all oxide failures contribute to the defect-related

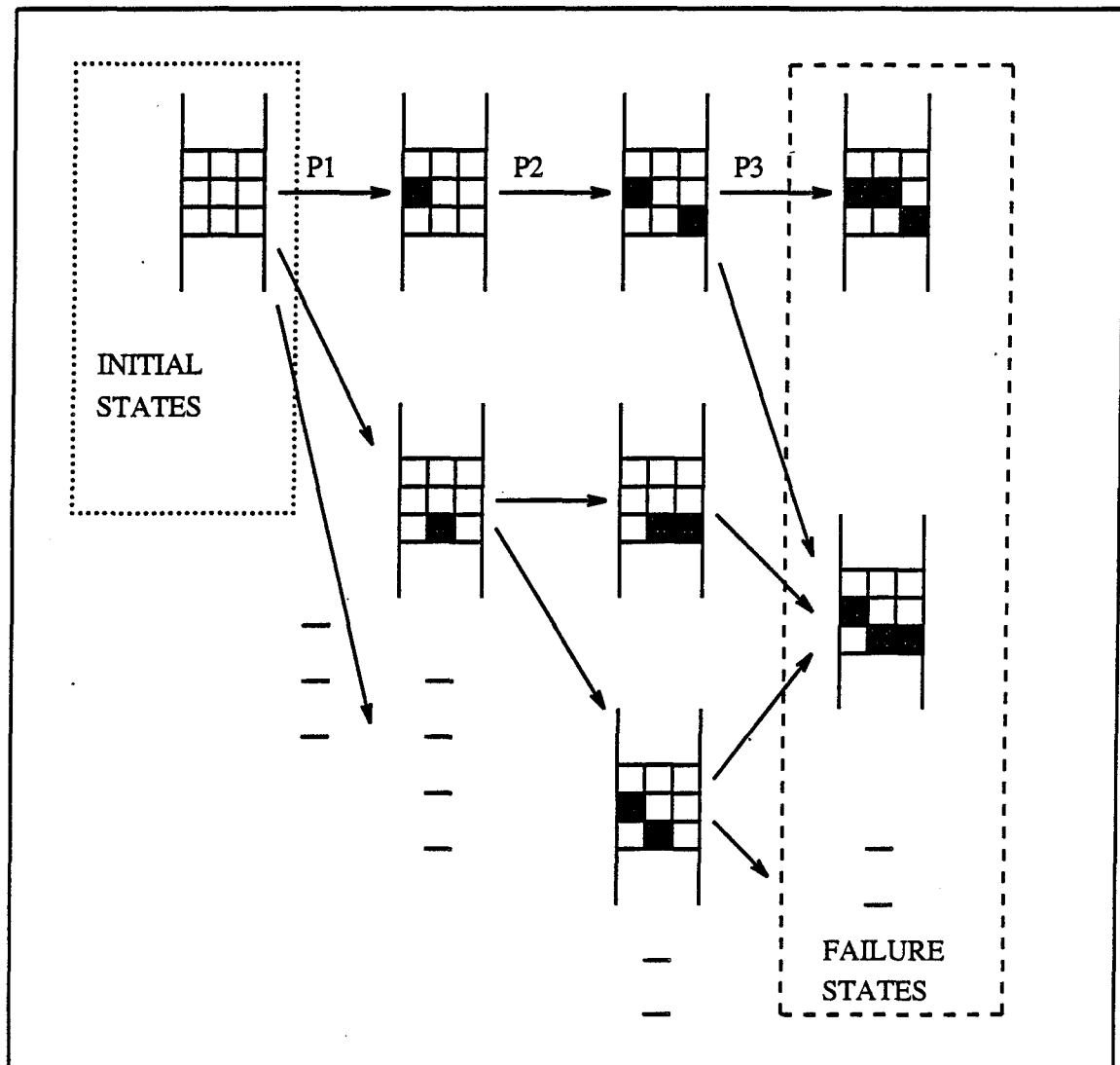


Figure 33: Electromigration wear-out process.

```

Calculate grain-removal probabilities and store it in LOOKUP_TABLE
For each time tick {
  Increment time by simulation resolution
  If (Switching event is observed from trace){
    Evaluate_Switching_Event() {
      Evaluate electromigration{
        For the node(matrix) corresponding to the detected event {
          For each grain in the matrix{
            Look up the removal probability
            from LOOKUP_TABLE(generated in initially)
          }
          Decision variable = random number between [0,1]
          If (Decision variable < removal probability) {
            Remove grain - Update matrix
            If void-path (failure) detected then Exit
            Recalculate removal probabilities{
              For each grain in the same
              column in the matrix{
                Recalculate removal probability
                (Compensate for increased current
                through, now, a thinner metal line)
              }
            }
            Update LOOKUP_TABLE
          }
        }
      }
    }
  }
  Evaluate other Wearouts{...}
  -
  Real_Time = Time_Stamp_On_Event)
  If failure detected {
    Record {
      1. Failure mode (e.g. electromigration)
      2. List of transition probabilities
         for the transition path that led to
         the failure
      3. Biasing parameters
      Exit
    }
  }
}

```

Figure 34: The electromigration-simulation algorithm.

thinning effect that cannot be screened at the burn-in stage.

In our wear-out analysis, the *thinning effect* of gate oxide and subsequent failure, due to the operational stress, is considered. The *thinning effect* of gate

oxide occurs from charge trapping. This failure model is commonly accepted as one of the major failure modes [40]. Charge trapping occurs when the device is active, i.e., when the electrical field is applied on the oxide. We determine when the device (gate) is active from the circuit simulation trace.

We simulate time dependent gate oxide breakdown as imposed by electrical fields from actual switching conditions (from running actual software on the chip) for each gate. An example of the gate oxide model is shown in Figure 35. As shown in Figure 35, the level of dielectric condition of an oxide is modeled by the number of elements in a stack. The oxide condition is nominal when the representative stack is full. The process of oxide breakdown is modeled by decreasing the effective thickness of oxide which is implemented by removing elements from the stack. An oxide breakdown occurs when the stack is empty.

A Monte Carlo analysis similar to that used for electromigration analysis is used to simulate the oxide-breakdown process. An element in the stack may be removed, for each clock period when the gate is active, based on a probability calculated from experimental data. The MTTF of a target oxide was determined for different operating variables in [40].

We also consider the variation in oxide quality that is introduced during the fabrication process due to uneven oxidation. At the beginning of a simulation run, to model the defect distribution variance, the number of elements initialized for each stack is perturbed by the percent-tolerance parameter for oxide thickness in fabrication data.

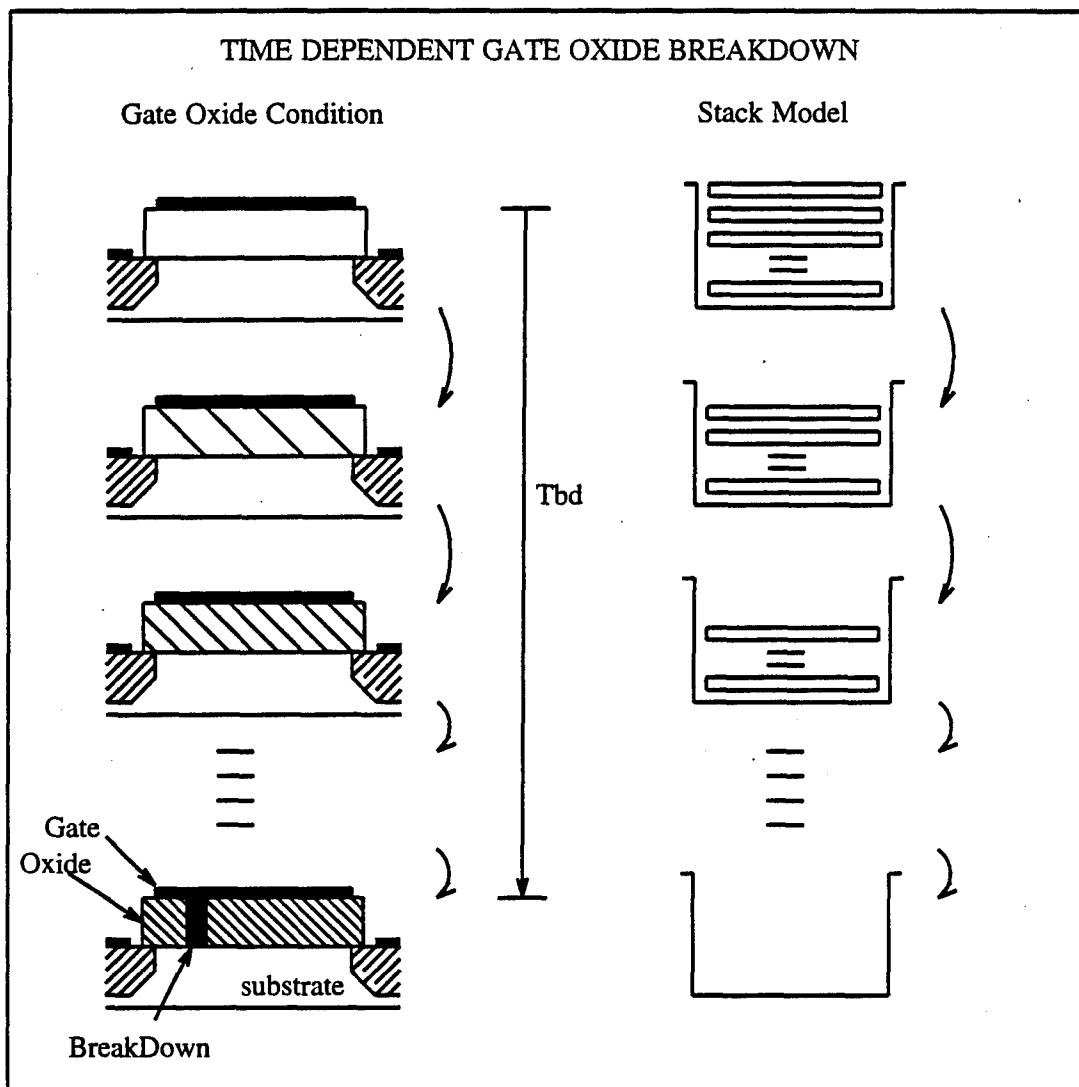


Figure 35: Modeled gate oxide.

If all elements in the stack are removed, i.e., oxide breakdown occurs, then a chip failure is assumed. The above wear-out processes are carried out in parallel for all the gates in the circuit.

The above analysis (electromigration and oxide breakdown) can be performed in parallel, under varying operating environments and fabrication technology parameters. In particular, operating voltage, temperature and the device

dimension can be varied and the reliability impact of reduced dimension and technology improvements can be quantified.

In the above Monte Carlo steps, the probability of having a grain removed from the matrix or having an element removed from the stack is extremely small to cause a failure within a reasonable simulation period. Thus we enhanced the Monte Carlo analysis approach to make automatic use of *importance sampling* to reduce the run lengths. The underlying theory of the method of importance sampling is described in the next section.

7.1.3. Importance sampling

The problem with direct simulation of device wear-out processes is the time required to perform the analysis, because the device failure, due to wear-out, occurs in the order of years if not in tens of years. Simulation of such a scenario is impossible with the capabilities of current computational resources. Thus, we employ importance sampling technique [51] to allow acceleration of the events causing the wear-out by biasing the related parameters to increase the chance of failure.

Importance sampling theory: Given a function $f(x)$, assume that we are interested in estimating the integral:

$$\int_0^1 f(x) dx$$

by taking a sample (x_1, x_2, \dots, x_N) from random variable x over the range $[0 < x < 1]$

1] and calculating the average of $f(x)$. The objective of importance sampling is to concentrate the distribution of the sample points on the parts of the interval that are of the highest *importance* instead of spreading them out evenly. Thus, instead of sampling following an uniform distribution, we introduce a sampling distribution with the density function $g(x)$:

$$\int_0^1 g(x) dx = 1.$$

In order not to bias the result, we compensate for the distortion by taking $f'(x) = \frac{f(x)}{g(x)}$ in place of $f(x)$ as our estimator of θ . The likelihood ratio is thus $\frac{f(x)}{g(x)}$, and the variance of this new unbiased estimator is

$$\int_0^1 \left(\frac{f(x)}{g(x)} - \theta \right)^2 g(x) dx.$$

For a minimum variance solution, $g(x)$ must be close to $\frac{f(x)}{\theta}$. However, obtaining the optimal $g(x)$ requires the knowledge of θ , which is unknown. The above considerations do provides some guideline for selecting $g(x)$. In particular, the shape of $g(x)$ should follow the shape of $f(x)$ as closely as possible.

Applications: Importance sampling is used to accelerate the wear-out scenario in our Monte Carlo simulations. The reliability measure we are interested in estimating is the mean-time-to-failure (MTTF).

In the electromigration analysis, the probability of grain removal in each switching event is very small since the frequency of the electromigration is in the order of years. Therefore, we focus on those metal lines that are most likely to fail. Metal lines having a higher rate of switching events have a higher chance of causing a failure and have their wear-out accelerated. We accomplish this effect by biasing the probability distribution of grain removal appropriately at each switching event. This procedure is referred to as transition probability mapping.

Transition probability mapping: At each switching event, we have the probability $P[state(i) \rightarrow state(i+1)]^{15}$ where $state(i)$ is a matrix pattern resulting from grain removal, due to a switching event i . Since the $P[state(i) \rightarrow state(i+1)]$ is very small (i.e., grain removal probability is small), no state transition is likely to occur within a reasonable simulation period. Hence, we use a new $p'[state(i) \rightarrow state(i+1)] (>> p)$ calculated as follows:

- 1) Maximum probability (P_{max}) of grain removal for all lines is calculated by taking the worst case condition of each line (i.e., single grain is left and, thus, high current flux flows through a smaller cross section), and calculating its removal probability for a switching event.
- 2) Minimum grain removal probability (P_{min}) is calculated in a similar way. All lines are considered, and removal probability while the metal line is fully intact.

¹⁵Capital letter P denotes probability, e.g., $P(event(i))$ is probability of event (i) occurring.

3) Project the original probability range to

$$P'[\text{state}(i) \rightarrow \text{state}(i+1)] \equiv ((P[\text{state}(i) \rightarrow \text{state}(i+1)] - P_{\min}) \times (\frac{P'_{\max} - P'_{\min}}{P_{\max}})) + P'_{\min}.$$

P'_{\max} and P'_{\min} are experimentally determined to make the state transition possible; yet, each transition is not so abrupt.

An example probability mapping for Monte Carlo simulation acceleration is illustrated in Figure 36. The original grain removal probabilities range from $6.20\text{E-}6(P_{\min})$ to $2.83\text{E-}5(P_{\max})$. With probabilities in this range, no grain removal is likely to occur within a reasonable simulation period. Thus we map the original probabilities to the $[2.0\text{E-}2, 5.0\text{E-}2]$ range to accelerate the wear-out process.

Unbiasing: Given a state transition path $\text{state}(i) \rightarrow \text{state}(i+1) \rightarrow \dots \rightarrow \text{state}(n-1) \rightarrow \text{state}(n)$ where $\text{state}(i)$ is clean state (i.e., no grain is removed) and $\text{state}(n)$ is failed state (refer to Figure 31), the probability of having a failure is

$$\prod P[\text{state}(i) \rightarrow \text{state}(i+1)], P[\text{state}(i+1) \rightarrow \text{state}(i+2)], \dots, P[\text{state}(n-1) \rightarrow \text{state}(n)].$$

The new biased path transition probability is

$$\prod P'[\text{state}(i) \rightarrow \text{state}(i+1)], P'[\text{state}(i+1) \rightarrow \text{state}(i+2)], \dots, P'[\text{state}(n-1) \rightarrow \text{state}(n)].$$

Thus the likelihood ratio of each Monte Carlo transition path is

$$L = \frac{\prod P[\text{state}(i) \rightarrow \text{state}(i+1)], P[\text{state}(i+1) \rightarrow \text{state}(i+2)], \dots, P[\text{state}(n-1) \rightarrow \text{state}(n)]}{\prod P'[\text{state}(i) \rightarrow \text{state}(i+1)], P'[\text{state}(i+1) \rightarrow \text{state}(i+2)], \dots, P'[\text{state}(n-1) \rightarrow \text{state}(n)]}.$$

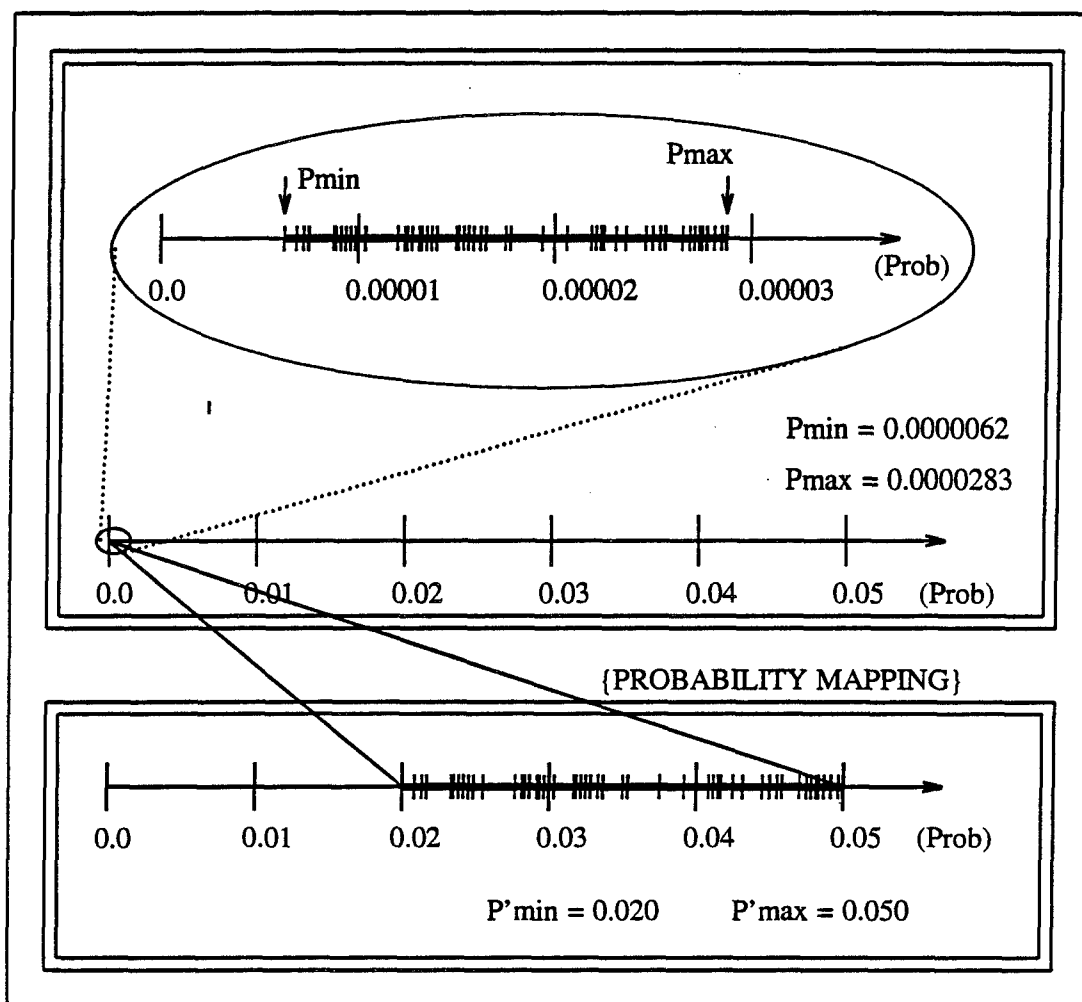


Figure 36: Transition probability mapping for wear-out acceleration.

So as not to bias the result, we compensate for the distortion due to the acceleration process. Each resulting accelerated time-to-failure is divided by the likelihood ratio of its unique transition path to obtain the desired real time-to-failure.

Acceleration for oxide breakdown: A similar importance sampling approach was taken for the oxide breakdown analysis. For the oxide breakdown analysis, probability of having an element removed from the stack is very small and the possi-

bility of having a failure within a reasonable simulation period is extremely low. Thus we choose a new probability Po' for removing elements instead of the original Po ($Po' \gg Po$) for a given gate oxide. The choice of Po 's was experimentally determined and was in the range [0.05 and 0.25].

7.2. Case Study

The HS1602 microprocessor is used to illustrate the wear-out simulation approach.

7.2.1. Experiment

First, the entire microprocessor was simulated at the switch level. The initialization phase of the microprocessor, consisting of a watchdog test, a parity test, an instruction set test, a RAM test, and a ROM sum test which ensures that all of the functional units are exercised, was simulated. The simulation included the processor accessing one external ROM for instructions and another external ROM for the initialization parameters. Arithmetic processing and address generation was also performed. Traces of events were collected from the simulation.

Using the switching activity information, the wear-out processes were simulated using Monte Carlo techniques. To model electromigration each metal line was represented by a 3 by 3 matrix of metal grains. We use the average metal grain size of 1/3 of the width of the metal line. The grains in the matrix were removed during the simulation based on a normal probability distribution and the current density calculated from the trace data. The probability of grain removal

was calculated from the physical experimental results reported in [36]. The oxide-breakdown was analyzed using stacks with a mean size of 32 elements. We ran initial experiments where the size of the stack was varied to determine the stack size to use. The stack size was chosen such that additional size would not enhance the accuracy of the result. First, 15 percent defect variance was introduced to the original stack size. The elements were taken out during the simulation based on a normal probability distribution.

The above analysis was performed under varying operating environments and fabrication technology parameters. The operating voltage and the device dimension were varied. Voltage levels tested were 3, 5 and 7 V. Different device scales ($2.1\ \mu$ $2.8\ \mu$ $3.5\ \mu$) were tested to study the reliability impact of reduced dimension. Other technology improvements such as metal lines with better conductance were also studied. A total of 2304 Monte Carlo simulations were performed. Increasing the number of Monte Carlo runs beyond this value did not significantly change the nature of our results.

7.2.2. Results

Reliability projections given by the wear-out analysis, at $T = 300^\circ K$ with a 5 V power supply, are shown in Figure 37. The figure shows the expected lifetime (y-axis) of the controller as a function of the overall average operational hours per day (x-axis) and the percent of actual time (line attribute) in use. For example (Figure 37: Example A), if the overall operational period is 12 hr/day and the actual usage of the controller is 75 percent of this time, then the expected

EXPECTED LIFE TIME OF THE CONTROLLER

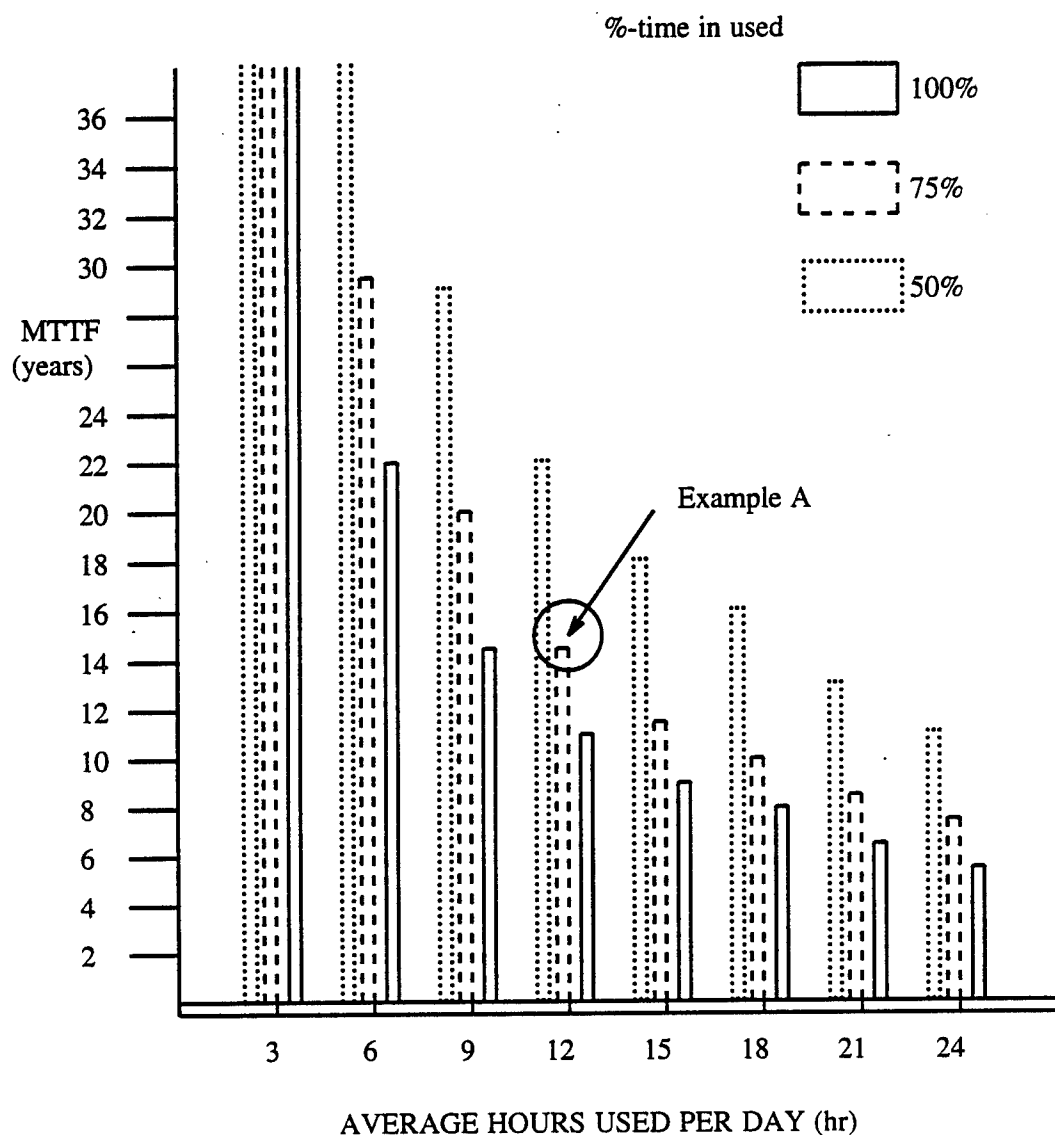


Figure 37: Expected lifetime of the target chip.

lifetime of the chip is about 14 years.

The time distributions of the failure points, for each device dimension, are shown in Figure 38. The circuit fabricated in 3.5- μ technology has an MTTF of 5.643 years and with a 90 percent confidence interval between 3.856 and 7.322

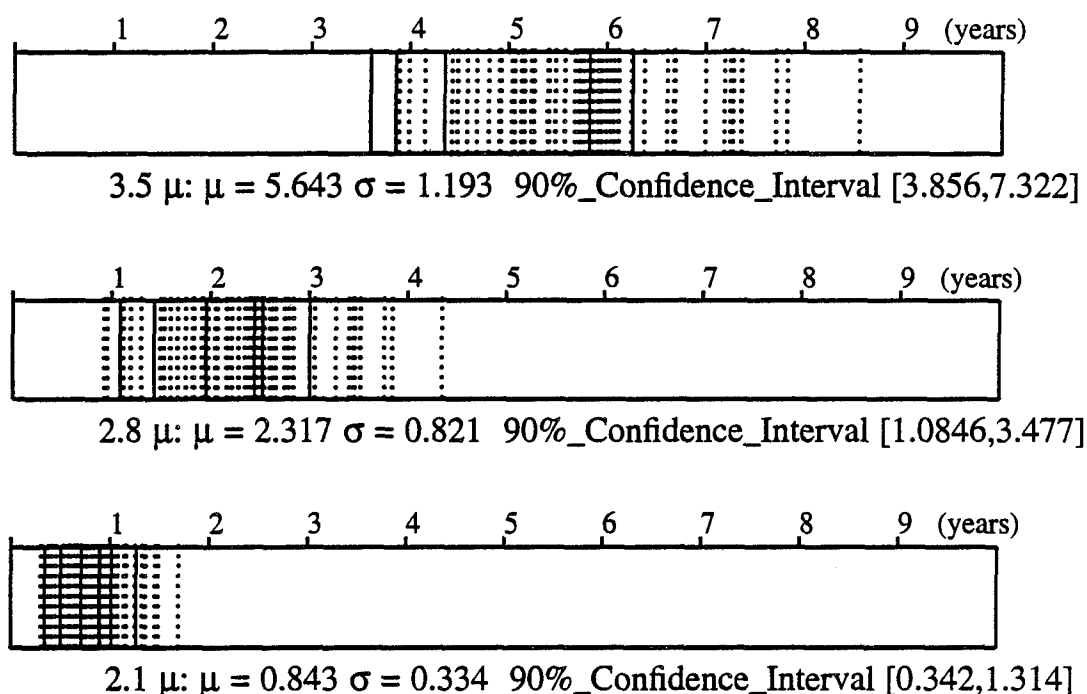


Figure 38: TTF distribution for different scale.

years. The dashed lines show the distribution of the failure points due to electromigration and the dotted line shows the distribution due to the oxide breakdown. The results show that most of the failures are due to electromigration. No more than 10 percent of the failures for the 3.5 μ chip were due to the oxide breakdown.

The results shown in Figure 39 give TTF distributions (y-axis) for operating voltage (x-axis). Ninety percent confidence limits are shown in dotted lines. A log-linear relation is observed between the TTF and operating voltage. Note that these are the results based on the assumption that all other operating variables (e.g., temperature) stays the same, which is generally not the case.

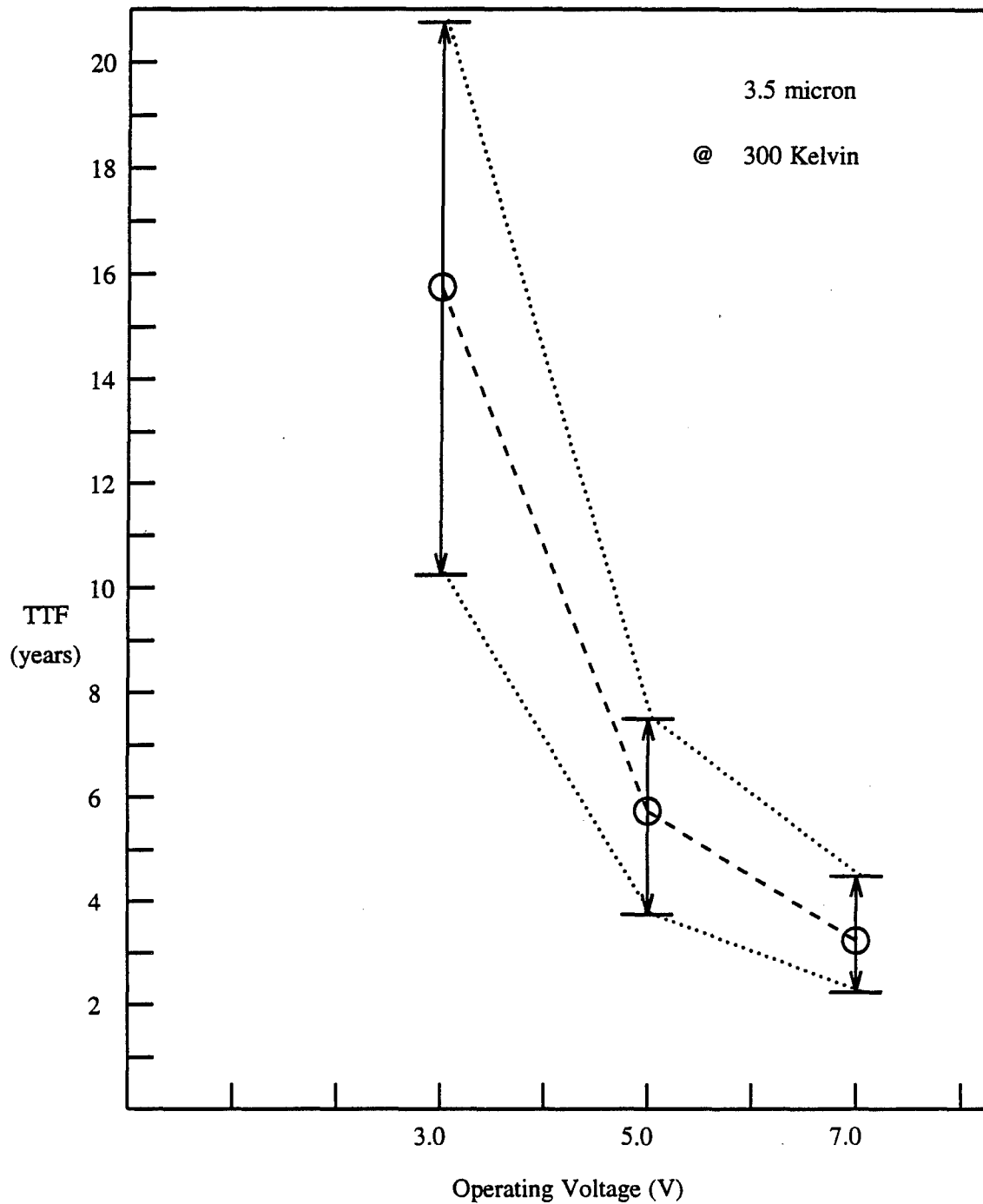


Figure 39: TTF distribution at different operating voltage.

The results of the simulation compare favorably with other reported experimental results. The reliability studies performed on Intel's 8086 microprocessor

are reported in [58]. Intel's 8086 microprocessor's fabrication conditions are approximately equivalent to those under which our target chip was fabricated. The report indicates that the chance of having a failure of the 8086 microprocessor at 343 Kelvin is about 0.097% at 1000 hr. This failure probability was calculated from the projection using the Arrhenius relationship from the accelerated testing data. The result of the Monte Carlo simulation at the same temperature is given in Figure 40. The figure shows cumulative density function of failure over time. The MTTF for this simulation run is 0.438 year and σ is 0.127 year. The cumulative density from the manufacturer's report is plotted with X on the same

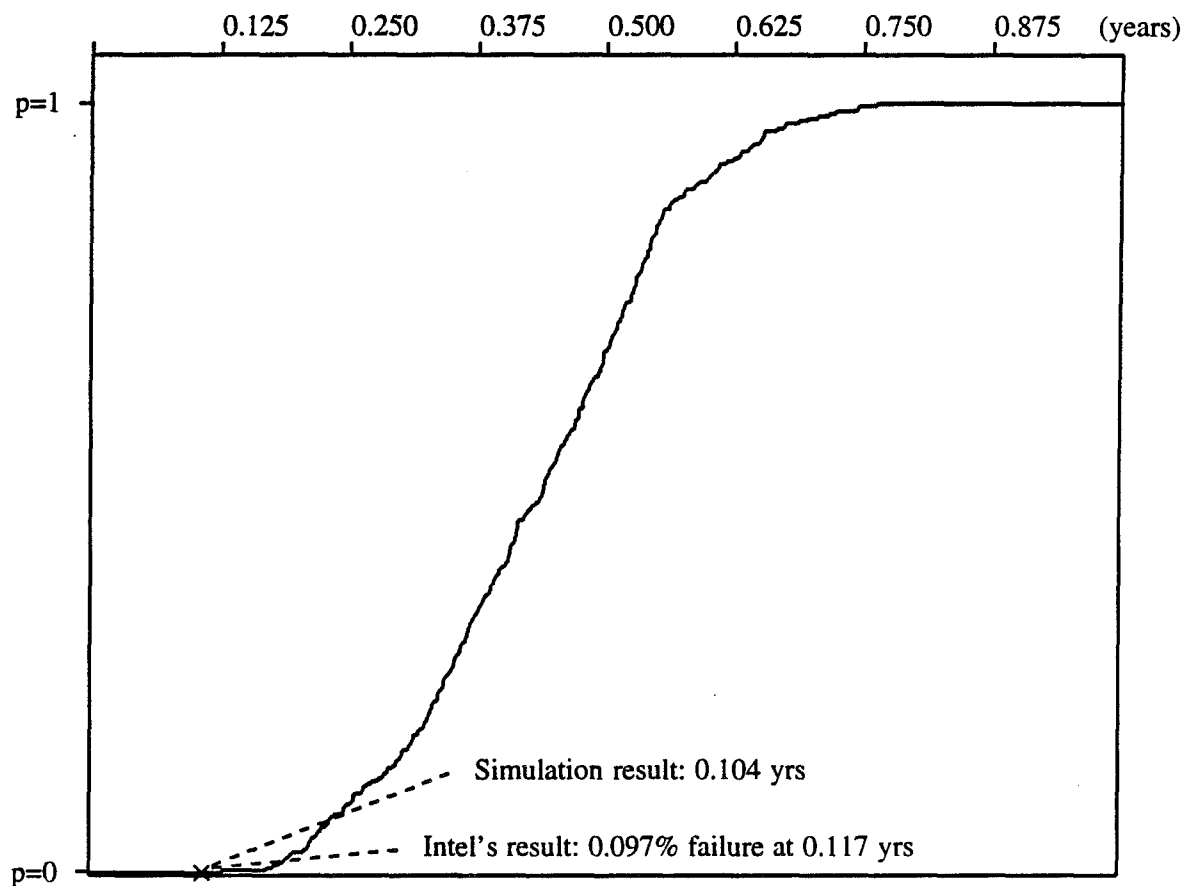


Figure 40: Cumulative density function(CDF) at 343 Kelvin.

figure (0.097 at $t=0.117$ year). A simple regression from the simulation result indicates that time for 0.10% ($\approx 0.097\%$ chance: manufacturer's report) to fail is about 0.104 year. Due to the high reliability of these VLSI systems, only the values at the tail region of the TTF distribution can be validated. However, it is important to note that we can generate the entire distribution. The percent difference between the two TTFs is about 11 percent. This difference is likely due to the consideration of only two wear-out/failure mechanisms, i.e., only the electromigration and oxide-breakdown mechanisms.

The simulation result also shows that there is a handful of detected metal lines that failed most frequently due to electromigration. To increase reliability, these metal lines can be physically modified (e.g., use thicker lines to route these wires) or an alternate logic design can be used to reduce the switching effects on these lines. Table 7 lists the lines that are most sensitive to failure. A large

Table 7: Most sensitive metal lines.

Location	node name	metal line length(μ)	Percentage
ALU	hs1602_m1_ca_al_m5_m2_u12_clkb	410	13.4%.
ALU	hs1602_m1_ca_al_m5_m2_u13_clkb	440	7.2%.
ALU	hs1602_m1_ca_al_m5_m2_u21_i2	680	5.8%.
ALU	hs1602_m1_ca_al_m5_m2_u22_i2	450	5.5%.
CON	hs1602_m1_c_u2	1460	5.1%
CON	hs1602_m1_c_u2_i8	620	3.9%
CON	hs1602_m1_c_u12	1220	3.6%
CON	hs1602_m1_c_u13	1580	3.6%
WDG	hs1602_m1_w_u1_clkb	340	2.4%
WDG	hs1602_m1_w_u1_clkb	340	1.2%
OTHER	-	-	52.3%
TOTAL	-	-	100%

share of the most sensitive lines are found in ALU and control units, because, relatively, the ALU is more highly exercised during the chip's operation, i.e., the lines in the ALU unit are switched on and off more frequently, because relatively longer lines are used in the control unit.

7.3. Discussion

This subsection described a simulation approach for reliability prediction of VLSI designs. The approach can effectively model the workload-reliability dependency of VLSI systems and identify fail-sensitive locations in the circuit at the design phase. The demonstrated Monte Carlo simulation technique allows the user to be able to observe dynamic processes of wear-out in the target chip. Currently, the approach supports simulation of electromigration and oxide-breakdown.

The approach was illustrated with a case study of the HS1602 microprocessor intended for control applications. The system under investigation was first simulated at the switch level and trace data on switching activity were collected. These data were then used along with the Monte Carlo simulation to model wear-out at the chip level.

CHAPTER 8.

CONCLUSIONS

The contributions of this thesis are divided into three subsections. First, the thesis introduces a hierarchical, mixed-mode simulation approach capable of injecting runtime transients and tracing their impact. The probability that a transient results in latch, pin, or functional errors can be determined. The approach also allows quantification of the impact of transient hardware errors at the software execution level. Second, to effectively evaluate long-term reliability, a method for predicting permanent faults in VLSI designs is developed. The method combines switch-level circuit simulation and device-level Monte Carlo simulation to achieve realistic reliability assessment. Third, the fault-sensitivity and reliability analysis methodologies are illustrated with the target systems, and the results are obtained.

8.1. Summaries

8.1.1. Transient fault sensitivity analysis

This thesis has presented a simulation approach which evaluates the fault sensitivity of a chip-level design. The approach effectively evaluates the fault-tolerance and the fault sensitivity of target systems. Faults are automatically injected in runtime at the device level, and their propagation and impact are monitored at the gate and function levels. A number of techniques for fault sensitivity analysis have been proposed and implemented in the mixed-mode

simulation approach. These include transient impact assessment on latch, pin and functional errors, external pin error distributions due to within-chip transients, and error propagation models to depict the dynamic behavior of latch errors.

The simulation analysis has been illustrated via a case study of the impact of transient faults on a microprocessor-based jet-engine controller. The simulation approach has been used to identify and isolate the critical fault propagation paths, the module most sensitive to fault propagation, and the module with the highest potential of causing external pin errors. The fault propagation path between the control unit and the watchdog unit is seen to be the most critical, indicating thereby that an increase in the fault tolerance of this link may significantly improve the system dependability. The watchdog unit has the highest potential for causing external pin errors. Of all the functional units, an error occurrence in the ALU is likely to lead to the largest number of latch errors.

A method to quantify the impact of low-level transients at the software execution is also presented. The software (program-flow level) upsets in VLSI systems have been analyzed using the experimental approach. Using test workloads, the types of upsets at the program-flow level which can result from fault propagation are determined. The mechanisms involved in internal propagation of latch errors and their effect at the software execution are modeled.

A Markov model was constructed from empirical data to describe the error propagation within the microprocessor and the subsequent impact at the program flow. The model was used to identify the functional unit most sensitive to error

propagation and the unit with the highest potential of causing software upsets. Several key error characteristics at the software level were observed.

Overall, there is 21.4 percent chance of having an upset. About 20 percent of all observed upsets are multiple in nature. Arithmetic operations in the application program are most susceptible to upsets. Forty and one-half percent of all observed upsets are contributed by a program section, which consists of, mostly, arithmetic instructions.

The coverage of the fault-tolerant design to single and multiple transients has been evaluated. The locations and the time points of the fault injections are selected so as to maximize the chance of channel errors. Specifically, faults are injected under conditions where critical communications were taking place within the dual system. The results show that the avionic controller had an estimated 100 percent coverage against single isolated transients, while approximately 12 percent of the multiple transients affected both channels.

The fault-dictionary approach to accelerate the mixed-mode fault-injection process is also presented. The gates around the fault-injection location are extracted and subcircuit consisting of these gates is formed. This subcircuit is exercised by exhaustively applying all input combinations while fault injection is performed. Faulty behavior at each of the subcircuit outputs is analyzed and recorded in a dictionary. The generated fault dictionary was used to inject, in runtime, the logical pattern/behavior of device level faults from a look-up table (dictionary) on a fast logic simulation. We find that 99.8 percent of the injected

faults have no effect on the system, i.e., no latch error would result from them. Multiple errors are less likely to occur (8 percent) but then can be a potentially serious problem since they can cause multiple failures.

8.1.2. Permanent fault wear-out simulation

The thesis has also described a simulation approach for reliability prediction of VLSI designs. The approach can effectively model the workload-reliability dependency of VLSI systems and identify fail-sensitive locations in the circuit at the design phase. The demonstrated Monte Carlo simulation technique allows the user to observe dynamic processes of wear-out in the target chip. Currently, the approach supports simulation of electromigration and oxide-breakdown. The approach was illustrated with a case study of the HS1602 microprocessor system. The system was first simulated at the switch level and trace data on switching activity were collected. These data were then used along with Monte Carlo simulation to model wear-out at the chip level.

8.2. Future Extensions

The need for innovative fault-sensitivity and reliability evaluation techniques will continue to grow as systems become more complex. Specifically, more methods are needed to perform yet faster fault simulations that can handle increasingly large VLSI systems. Also, to study the effects of device-level faults on software under realistic operating conditions, low-level wear-out analysis must be integrated into the fault-injection simulation environment. One of the important challenges that lie ahead is integrating fault/failure analysis to effectively evaluate

alternatives in design tactics and to aid in the synthesis of architectures that meet required fault-tolerant specifications.

REFERENCES

- [1] H. Ball and F. Hardy, "Effects and detection of intermittent failures in digital systems," 1969 FJCC, AFIPS Conference Proceedings, vol. 35, pp. 329-335.
- [2] R. K. Iyer and D. J. Rossetti, "A measurement-based model for workload dependence of CPU errors," *IEEE Transactions on Computers*, vol. C-35, pp. 511-519, June 1986.
- [3] T. C. May and M. H. Woods, "Alpha-particle-induced soft errors in dynamic memories," *IEEE Transactions on Electron Devices*, vol. ED-26, pp. 2-9, January 1979.
- [4] R. J. McPartland, "Circuit simulations of alpha-particle-induced soft errors in dynamic RAM's," *IEEE Journal of Solid-State Circuits*, vol. SC-16, pp. 31-34, February 1981.
- [5] R. Johnson, S. Diehl-Nagle and J. Hauser, "Simulation approach for modeling single event upsets on advanced CMOS SRAMS," *IEEE Transactions on Nuclear Science*, vol. NS-32, pp. 4122-4127, December 1985.
- [6] G. C. Messenger, "Collection of charge on junction nodes from ion tracks," *IEEE Transactions on Nuclear Science*, vol. NS-29, pp. 2024-2031, December 1982.
- [7] J. Lala, "Fault detection isolation and reconfiguration in FTMP: Methods and experimental results," The 5th AIAA/IEEE Digital Avionics Systems Conference (DASC), pp. 21.3.1-21.3.9, 1983.
- [8] K. G. Shin and Y.H. Lee, "Error detection process - model, design, and its impact on computer performance," *IEEE Transactions on Computers*, vol. C-33, pp. 529-540, June 1984.
- [9] K. G. Shin and Y.H. Lee, "Measurements of fault latency: methodology and experimental results," Technical Report CRL-TR-45-84, Computing Research Laboratory, University of Michigan, Ann Arbor, 1984.

- [10] J. Arlat, Y. Crouzet and J. Laprie, "Fault-injection for dependability validation," LAAS Research Report no. 88-363, December 1988.
- [11] B. Courtois, "Some results about the efficiency of simple mechanisms for the detection of microcomputer malfunctions," Digest, FTCS-9, The Ninth International Symposium on Fault Tolerant Computing, pp. 71-74, June 1979.
- [12] R. E. Glaser and G. M. Masson, "Transient upsets in microprocessor controllers," Digest, FTCS-11, The Eleventh International Symposium on Fault Tolerant Computing, pp. 165-167, 1981.
- [13] M. E. Schmid, R. L. Trapp, A. E. Davidoff and G. M. Masson, "Upset exposure by means of abstraction verification," Digest, FTCS-12, The Eleventh International Symposium on Fault Tolerant Computing, pp. 237-244, 1982.
- [14] R. Koga, W. A. Kolasinski, and M. T. Marra, "Techniques of microprocessor testing and SEU-rate prediction," *IEEE Transactions on Nuclear Science*, vol. NS-32, pp. 4219-4224, December 1985.
- [15] J. Sosnowski, "Evaluation of transient hazards in microprocessor controllers," Digest, FTCS-16, The Sixteenth International Symposium on Fault Tolerant Computing, pp. 364-369, 1986.
- [16] R. Chillarege and R. Iyer, "Measurement-based analysis of error latency," *IEEE Transaction on Computers*, vol. C-36, no. 5, May 1987.
- [17] J. Cusick, R. Koga, W. A. Kolasinski, and C. King, "SEU vulnerability of the Zilog Z-80 and NSC-800 microprocessors," *IEEE Transactions on Nuclear Science*, vol. NS-32, pp. 4206-4211, December 1985.
- [18] J. Karlsson, U. Gunneflo, and J. Torin, "The effects of heavy-ion induced single event upsets in the MC6809E microprocessor," Proceedings, 4th International Conference on Fault-Tolerant Computing Systems, GI/ITG/GMA, Baden, W. Germany, 1989.

- [19] U. Gunneflo, J. Karlsson, and J. Torin, "Evaluation of error detection schemes using fault injection by heavy-ion radiation," Digest, FTCS-19, pp. 340-347, 1989.
- [20] D. Lomelino and R. Iyer, "Error propagation in a digital avionic processor: a simulation-based study," Proceedings, Real Time Systems Symposium, pp. 218-225, Dec. 1986.
- [21] P. Duba and R. Iyer, "Transient fault behavior in a microprocessor: a case study," 1988 ICCD Proceedings, October 1988.
- [22] G. Choi, R. Iyer, R. Saleh and V. Carreno, "Fault behavior model for an avionic microprocessor," Proceedings, International Working Conference on Dependable Computing For Critical Applications, Santa Barbara, CA, August 1989.
- [23] E. W. Czeck, "On the prediction of fault behavior based on workload," Ph.D. Dissertation, Dept. of Electrical and Computer Engineering, Carnegie Mellon University, April 19, 1991.
- [24] R. Iyer and D. Tang, "Experimental analysis in computer system dependability," *Fault-Tolerant Computing*, D. K. Pradhan, Ed.: Prentice Hall, 2nd ed., 1994.
- [25] F. Yang, "Simulation of faults causing analog behavior in digital circuits," Ph.D. thesis, University of Illinois, 1992.
- [26] J. McGough and F. Swern, "Measurement of fault latency in a digital avionic mini processor," NASA Contract Report 3462, Washington, DC 1981.
- [27] A. Dupuy, J. Schwartz, Y. Yemini, and D. Bacon, "NEST: A network simulation and prototyping testbed," Communications of the ACM, vol. 33, no. 10, pp. 64-74, October 1988.
- [28] K. Goswami and R. Iyer, "DEPEND: A design environment for prediction and evaluation of system dependability," Proceedings, 9th Digital Avionics Systems Conference, October 1990.

- [29] J. Clark and D. Pradhan, "REACT: A synthesis and evaluation tool for fault-tolerant multiprocessor architectures," *Proceedings, Annual Reliability and Maintainability Symposium*, pp. 428--435, 1993.
- [30] M. Woods, "MOS VLSI reliability and yield trends," *Proceedings of the IEEE*, vol. 74, no. 12, pp. 1715-1729, December 1986.
- [31] J. Black, "Electromigration failure modes in aluminum metallization for semiconductor devices," *Proceedings of the IEEE*, vol. 57, no. 9, pp. 1587-1593, 1969.
- [32] C. Hu, P. Ko, P. Lee, N. Cheung, and B. Liew, "IC Reliability prediction," SRC TECHCON-88, Dallas TX, pp. 240-243, October 88.
- [33] J. McPherson, "Stress dependent activation energy," *IEEE Proceedings, IRPS*, pp. 12-18, April 1987.
- [34] D. Frost, K. Poole, "RELIANT: A reliability analysis tool for VLSI interconnects," *IEEE Journal of Solid-State Circuits*, vol. 24, no. 2, pp. 458-462, April 1989.
- [35] J. Harrison, "On extrapolation from accelerated test MTTF to operating condition MTTF for electromigration failures," SRC TECHCON-88, Dallas TX, pp. 240-243, October 88.
- [36] D. LaCombe and E. Parks, "The distribution of electromigration failures," *IEEE Proceedings, IRPS*, pp. 1-6, April 1986.
- [37] L. Brooke, "Pulsed current electromigration failure model," *IEEE Proceedings, IRPS*, pp. 136-144, April 1987.
- [38] P. Marcoux, P. Merchant, V. Naroditsky, and W. Rehder, "A new 2D simulation model of electromigration," *Hewlett-Packard Journal*, June 1989.
- [39] I. Chen and C. Hu, "Accelerated testing of time-dependent breakdown of silicon dioxide," *IEEE Electron Device Letters*, vol. EDL-8, no. 4, pp. 140-142, April 1987.

- [40] J. Lee, I. Chen, and C. Hu, "Statistical modeling of silicon dioxide reliability," *IEEE Proceedings, IRPS*, pp. 131-138, 1988.
- [41] B. Ricco, M. Azbel, and M. Bordsky, "Novel mechanism for tunneling and breakdown of thin silicon dioxide films," *Physics Review Letter*, vol. 51, no. 19, pp. 1795, 1983.
- [42] M. Cortes and R. Iyer, "Device failures and system activity: a thermal effects model," *FTCS-14*, 1984.
- [43] R. Iyer and D. Rossetti, "A measurement-based model for workload dependence of CPU errors," *IEEE Transactions on Computers*, vol. C-35, pp. 511-519, June 1986.
- [44] Motorola Corporation, *MC68000 Programmer's reference manual*, Englewood Cliffs: N.J., Prentice-Hall, 1986.
- [45] R. A. Saleh, "Nonlinear relaxation algorithms for circuit simulation," Memorandum no. UCB/ERL M87/21, Electronics Research Laboratory, University of California, Berkeley, 1987.
- [46] J. Stephen, T. Sanderson, D. Mapper, J. Farren, R. Harboe-Sorensen, and L. Adams, "A comparison of heavy ion sources used in cosmic ray simulation studies of VLSI circuits," *IEEE Transactions on Nuclear Science*, vol. NS-31, no. 6, December 1984.
- [47] D. Nichols, W. Price, W. Kolasinski, R. Koga, J. Pickel, J. Blandford, Jr., and A. Waskiewicz, "Trends in part susceptibility to single event upset," *IEEE Transactions on Nuclear Science*, vol. NS-32, no. 6, December 1985.
- [48] M. Abramovici, M. Breuer, and A. Friedman, "Digital systems testing and testable design," *Computer Science Press*, 1990.
- [49] Richard J. Evans, "Detecting bridging faults in CMOS circuits," Ph.D. Dissertation, University of Oxford, 1991.

- [50] L. W. Nagel, "SPICE2: a computer program to simulate semiconductor circuits," ERL Memo ERL-M520, University of California, Berkeley, May 1975.
- [51] J. Hammersley and D. Handscomb, *Monte Carlo methods*, Methuen, London, 1964.
- [52] D. Saab, R. Mueller, D. Blaauw, J. Abrahm, and J. Rahmeh, "Hierarchical multi-level fault simulation of large systems," *JETTA Journal of Electric Testing: Theory and Applications*, no. 2, pp. 139-149, vol. 1, March, 1990.
- [53] E. Ulrich and T. Baker "Concurrent simulation of nearly identical digital networks," *Computer*, vol. 77, no. 4, pp.39-44, April 1974.
- [54] M. Rimen and J. Ohlsson, "A study of the error behavior of a 32-bit RISC subjected to simulated transient fault injection," *International Test Conference*, pp. 696-704, 1992.
- [55] H. Cha, E. Rudnick, G. Choi, J. Patel, and R. Iyer, "A fast and accurate transient fault simulation environment," *Digest, 23rd Int. Symp. Fault-Tolerant Computing*, pp. 310-319, June 1993.
- [56] G. Ries, G. Choi, and R. Iyer, "Transient fault modeling," *Digest, 24th Int. Symp. Fault-Tolerant Computing*, June 1994.
- [57] P. Shahabuddin, V. Nicola, P. Heidelberger, A. Goyal, and P. Glynn, "Variance reduction in mean time to failure simulations," *Proceedings of the 1988 Winter Simulation Conference*, pp. 491-499.
- [58] Intel, "iAPX 86, 88 microprocessor family," *Intel Reliability Report RR-27*, Intel Corporation, Febuary 1981.
- [59] Y.-H. Shih, Y. Leblebici and S. M. Kang, "ILLIADS: a fast timing and reliability simulator for digital MOS circuits," *IEEE Transactions Computer-Aided Design*, vol. 12, no. 9, pp. 1387-1402, Sept. 1993.

VITA

Gwan Choi received the B.S. Degree in Computer Engineering in 1989 from the University of Illinois and the M.S. Degree in Electrical Engineering in 1990 from the University of Illinois. He has held summer positions at the Cray Research Inc. and Tandem Computer Inc. and has been a research assistant at the University of Illinois since 1988. He has published 19 papers in peer-reviewed conferences and journals. He is a member of the IEEE. Upon completion of his Ph.D. work, he will join Texas A&M University as a faculty member in the Department of Electrical Engineering.